Sônia Leila Fernandes Silva

ConTOM
Um Sistema de Consultas Gráficas a um Banco de Dados Temporal Orientado a Objetos

Campina Grande
1995

Sônia Leila Fernandes Silva

ConTOM
Um Sistema de Consultas Gráficas a um Banco de Dados Temporal Orientado a Objetos

Dissertação apresentada ao Curso de Mestrado em Informática da Universidade Federal da Paraíba, como requisito parcial à obtenção do título de Mestre em Informática.

Área de Concentração: Banco de Dados

Orientador: Prof. Ulrich Schiel
Universidade Federal da Paraíba

Campina Grande
Universidade Federal da Paraíba
1995

S586c Silva, Sônia Leila Fernandes.
ConTOM : um sistema de consultas gráficas a um banco de dados temporal orientado a objetos / Sônia Leila Fernandes Silva. - Campina Grande : 1995.
135 f.

Dissertação (Mestrado em Informática) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1995.
"Orientação : Prof. Dr. Ulrich Schiel".
Referências.

1. Banco de Dados. 2. Consultas Gráficas. 3. ConTOM. 4. Interface Gráfica. 5. Dissertação - Informática. I. Schiel, Ulrich. II. Universidade Federal da Paraíba - Campina Grande (PB) III. Título

CDU 004.65(043)

ConTOM - Um Sistema de Consultas Gráficas a um Banco de Dados Temporal Orientado a Objetos

Sônia Leila Fernandes Silva

Dissertação aprovada em 21.02.1995

Ulrich Schiel, Dr.
Presidente

Maria de Fátima Q. V. Turnell, Ph. D
Componente da Banca

Décio Fonseca, Dr.
Componente da Banca

Slimane Hammoudi, Dr.
Componente da Banca

Campina Grande, 21 de fevereiro de 1995

A meus pais

Agradecimentos

A todos que contribuíram diretamente ou indiretamente para a realização deste trabalho. Gostaria de agradecer especialmente a algumas pessoas em particular:

- Aos meu pais, pelo apoio, carinho, estímulo e dedicação que sempre me proporcionarão.
- Ao Prof. Ulrich Schiel, pela orientação competente, experiência, disponibilidade e, principalmente, pelo estímulo e paciência nos momentos difíceis do trabalho. Sua colaboração foi de vital importância para o meu enriquecimento pessoal e profissional.
- Á Prof. Fátima Turnell, a primeira pessoa que me impulsionou para o desenvolvimento deste trabalho.
- Ao Prof. Slimane Hammoudi pela organização deste trabalho.
- Á minha amiga-irmã e conterrânea Egídia, uma grande incentivadora para eu poder continuar o trabalho.
- Ao Dr. Alberto, pelo tratamento eficaz da minha saúde.
- Aos Professores e Funcionários do Departamento de Sistemas, pelo apoio que viabilizou esta pesquisa, em particular, ao Prof. Peter, Aninha, Lilian, Alberto, Zeneide e Gustavo.
- Ás pessoas que durante todos estes anos me ajudaram, na colaboração técnica, troca de idéias, amizade, companheirismo e afeto. Em particular:
  - Aos Professores Edilson, Cláudio Baptista, Bernardo Lula, Joseluce, Hattori, Hélio Menezes e Tota.
  - Aos colegas dos papos noturnos no laboratório: Washington, Haroldo, Belo, Reginaldo, Robson, André Felipe, Edilson & Stanley (pelas caronas e água de côco).
  - Aos amigos próximos: Walfredo & Izabel, Frazão & Marília, Cristina, Jane, Iasnaia, Sebastian e Cortêz.
  - Aos amigos distantes: Diana, Luiz Antônio (Brasília), Fernando "Mandi", Fernandinho (Lagartixa) & Elmar, Lauro, Luiz Maurício, Ingrid & Léo e Akemi.
  - Aos novos colegas do mestrado e do laboratório: Gilson, Edberto, Ricardo, Kátia, Kíssia, Norma, Vera, Michelle, Ismênia & Álvaro, Almir, Vítor, Carlos, Adna, Marcos, Lavoisier, Augusto, Sandro, Vladimir, Michelle, Ernandes, as meninas da cantina, etc, etc (desculpe se eu esqueci algum nome, como já dizia meu amigo Walfredo, a memória humana é muito fraca.....)

Lista de Figuras

Lista de Figuras

Lista de Figuras

Lista de Tabelas

Sumário

Resumo

Este trabalho apresenta o Sistema de Consultas Gráficas a Banco de Dados - ConTOM (Consultas TOM), considerando como aspectos relevantes: a especificação visual do esquema conceitual dentro de uma perspectiva modular, e o processo de especificação de uma consulta, constituído de operadores gráficos que são executados através da manipulação direta sobre os objetos do esquema conceitual. ConTOM explora integralmente a representação visual, reduzindo a um mínimo necessário dados textuais. Os operadores gráficos são divididos em operadores básicos e temporais. Estes últimos manipulam BD's temporais, integrando à representação visual a especificação de consultas temporais. O resultado da consulta é um subesquema com a mesma estrutura do esquema original, podendo ser reutilizado. O trabalho também apresenta: uma formalização do esquema conceitual e de todos os operadores gráficos, um estudo comparativo entre ConTOM e outras interfaces gráficas, e o projeto de interface da ferramenta ConTOM.

Capítulo 1

Introdução

1.1. A Expansão das Interfaces de Consulta

Durante os últimos anos, dentro da área de Banco de Dados (BD's), mais especificamente na área de **Consultas,** diversas pesquisas têm sido desenvolvidas em torno da criação de ferramentas que proporcionem ao usuário final, uma interface com representação visual, que seja de fácil manipulação e que forneça um acesso simples e rápido a grandes volumes de dados.

Algumas interfaces visuais de consulta foram introduzidas [McDonald75], [Wong82], [Bryce86], [Mark87], [Ty88], [KimHJ88], [Angelaccio90], [Schneider91], [Paredaens92], [Ramos92], [Mohan93], [Oliveira93], [Sockut93], [Vadaparty93], [Weiland93], como alternativas para suprir as deficiências das linguagens de consulta não procedurais, tais como **QUEL** ou **SQL**, considerando dois aspectos: a interação entre o usuário e uma base de dados, e as limitações dos modelos de dados em que elas se baseiam diante de aplicações mais complexas. As linguagens de consulta convencionais, embora não sendo procedurais e possam ser utilizadas de forma eficiente no sentido de formular consultas *ad hoc,* acabaram sendo restritas, na sua maioria, a usuários especialistas na área [Ramos92], não sendo muito compreensíveis e amplamente utilizadas por usuários novos ou não especialistas que, além de passarem por um período de aprendizado até alcançarem o domínio pleno da linguagem, são forçados a conhecerem alguns conceitos de álgebra relacional, cálculo do predicado, teoria do conjunto e junção.

Além das limitações das linguagens convencionais discutidas acima, existem outros fatores que serviram como estímulo à adoção de técnicas visuais nas interfaces de consulta modernas, e sua ampla difusão:

- A utilização de imagens, ao invés de linguagem textual, se tornou crucial dentro da interação homem-máquina. Segundo [Catarci93], a abordagem visual atrai a atenção do usuário e estimula-o a explorar todas as funcionalidades disponíveis no sistema.

- *Workstations* com *display bit mapped,* dispositivos de apontamento (*mouse),* ícones, janelas, menus *pop-up*, enfim, uma gama de facilidades gráficas, a baixo custo, se tornaram mais populares entre a comunidade de usuários de computação.

- O surgimento dos modelos de dados semânticos e orientados a objeto, com maior poder de expressividade na representação conceitual dos dados extraídos do mundo "real".

- O próprio processo de formulação de consulta é totalmente adequado para a utilização de técnicas visuais.

- O crescimento da necessidade de acessar BD's por usuários não especializados.

1.2. Algumas Interfaces Gráficas

O nosso sistema gráfico combina a funcionalidade de diversas interfaces existentes, acrescentado de algumas características adicionais. Faremos uma breve descrição sobre algumas interfaces gráficas de consulta.

O trabalho de [Bryce86] apresenta uma interface gráfica para o modelo semântico IFO, o sistema ***SNAP***. Toda a manipulação é feita sobre os esquemas conceituais. O esquema é representado por um grafo que possui figuras geométricas distintas e são empregadas para representar papéis e relacionamentos diferenciados. Uma característica interessante do sistema é suportar visões de esquema em uma forma modular através de uma variedade de mecanismos de abstração.

G-OQL [Ty88] é uma interface gráfica para a linguagem de consulta orientada a objeto ***OQL*** [Alashqur89]. Uma consulta OQL é considerada uma função, que aplicada ao BD, retorna um sub-BD com a mesma estrutura do BD original. A estrutura de um esquema de BD em OQL consiste basicamente de classes e associações entre estas classes que são representados em forma de diagrama. A interação pela G-OQL se realiza através da seleção das classes dentro de um diagrama, e na entrada de operadores de associação entre as classes selecionadas.

O sistema ***PICASSO*** [KimHJ88] apresenta uma linguagem de consulta gráfica para BD's sob o modelo de Relação Universal. A formulação da consulta é feita sobre hipergrafos utilizando o *mouse* com três botões de controle, o botão da esquerda é usado para seleção de atributos, o do centro para elaboração de predicados, e o da direita para escolha de opções no menu de processamento de consultas. Uma ferramenta de apoio permite a navegação sobre o resultado da consulta e a construção de consultas complexas a partir do resultado de consultas simples.

A interface do sistema ***GOOD*** [Paredaens92] apresenta três componentes básicos: esquema conceitual, esquema de consulta e instâncias de BD's. Os três componentes estão dentro de uma mesma representação gráfica, denominados grafos de esquema, grafo de consulta e grafo de instância, respectivamente. O grafo de consulta é abstraído a partir da aplicação de uma primitiva sobre o grafo de esquema, através da duplicação e identificação de nodos e ligações componentes deste grafo.

Em [Ramos92], pode-se observar uma proposta de interface gráfica para banco de dados relacional: a interface ***IQL***. Além de fornecer uma interação gráfica durante a formulação de consultas SQL, fornece mecanismos de otimização das mesmas. Usuários tem a seu dispor uma variedade de mecanismos de abstração tais como denominação de consultas e o uso de parâmetros, permitindo uma reutilização modular e genérica, possibilitando, desta forma, uma abordagem de orientação a objeto.

Uma nova interface para SGBDOO's (Sistemas de Gerenciamento de Banco de Dados Orientados a Objetos) foi descrito em [Oliveira93]. O sistema ***GOODIES*** vem com uma abordagem inteiramente nova para desenvolvimento de interfaces para BD's: não vem acoplado a um SGBDOO específico, ele se fundamenta nas principais características de um modelo de dados orientado a objetos e foi construído de modo independente de uma implementação específica destas características. As informações sobre esquemas e dados em *GOODIES*, são apresentadas textualmente, em janelas padronizadas, com facilidades de organização, navegação e consulta sobre as mesmas.

Enfim, o trabalho descrito em [Weiland93] propõe uma interface gráfica de uma forma mais intuitiva e acessível ao usuário, baseado nos conceitos hierárquicos de agregação e generalização, os quais contêm nodos representativos e manipuláveis. O protótipo da interface foi avaliado experimentalmente em comparação a uma interface de consulta textual; a interface gráfica obteve maior sucesso, principalmente entre usuários casuais ou com pouco treinamento em BD's.

1.3. Descrição Geral da Dissertação

O trabalho proposto apresenta **ConTOM (Consultas TOM)** - um sistema gráfico de consultas sob o modelo de dados orientado a objetos **TOM** *(Temporal Object Model)* [Schiel 91, David92]. ConTOM é constituído de operadores gráficos, de forma a expressar as consultas, e uma variedade de funcionalidades que facilitam a interação com o usuário. A especificação a nível de representação visual de um esquema conceitual, utilizando o modelo TOM, é denominada, em ConTOM, de **Esquema Conceitual Gráfico (ECG).**

ConTOM fornece a geração de um ECG de uma forma totalmente interativa, utilizando uma representação baseada no trabalho descrito em [Schiel84], que capacita o usuário a projetar e visualizar

um esquema de uma forma **modular**, através de mecanismos de abstração. A funcionalidade de um esquema com tal característica foi estimulada segundo a idéia de que não houve bastante progresso em fornecer acesso e manipulação conveniente aos esquemas de BD's mais complexos [Bryce86].

Uma consulta em ConTOM é expressa visualmente através da manipulação direta sobre qualquer componente de um ECG. A elaboração de uma consulta implica na criação de esquemas intermediários, obtidos pelos operadores gráficos sobre as classes componentes do esquema, denominados **Operadores Gráficos de Consulta (OGC)**, até a formulação completa da consulta, gerando um sub-esquema do ECG, denominado **Esquema de Resposta (ER).** O ER especifica qual a porção do BD a ser recuperado pela consulta e possui a mesma estrutura do ECG original, podendo ser salvo como uma visão e reutilizado para outras consultas posteriores.

1.4. Objetivos da Dissertação

Em [Catarci93], é mostrado como um BD e uma linguagem de consulta são formalmente definidos em termos de modelo de dados e um conjunto de operadores, respectivamente. Um modelo de dados é expresso em termos de uma ou mais representações, o mesmo se aplica a uma linguagem de consulta, onde os operadores possuem uma representação. Dado o interesse cada vez maior na pesquisa sobre a interação homem-máquina, houve uma crescente preocupação na representação visual, tanto dos modelos de dados, quanto às consultas.

Um outro aspecto, que diz respeito às interfaces gráficas de consulta, é a sua vantagem em relação às interfaces orientadas a comando, comprovada através de um estudo empírico realizado por [Weiland93], onde são avaliados experimentalmente diversos tipos de consulta, associados aos fatores tempo de resposta e taxa de erros (fatores relevantes de motivação do usuário em utilizar uma aplicação no computador) aplicados em um protótipo de uma interface gráfica e em uma interface baseada em comandos. Na análise dos dados obtidos, o resultado final mostrou que os fatores são menores na interface gráfica (as diferenças de tempo de resposta são estatisticamente insignificantes, porém as taxas de erro não são).

Motivado pelo trabalho realizado por [Weiland93] e [Catarci93], existem os seguintes objetivos que são fundamentais neste trabalho:

1. Tornar a representação tão importante quanto o modelo sob a qual ela se fundamenta, fornecendo uma representação gráfica de um modelo orientado a objetos, e de um conjunto de primitivas gráficas sobre tal representação, além de um suporte que permite a divisão da representação gráfica em módulos, dependendo da sua complexidade.

2. Adaptar a facilidade de consulta ao poder semântico do modelo de dados.

3. Permitir ao usuário ter acesso a uma base de dados através da formulação e manipulação de consultas no **mesmo** ambiente de classes e relacionamentos do esquema conceitual do BD, por meio de um reconhecimento visual dos objetos de interesse de uma consulta, sem haver a necessidade de um domínio da sintaxe de uma linguagem de consulta como SQL, por exemplo.

1.5. Características Básicas do ConTOM

As principais características do sistema ConTOM, são as seguintes:

- Os componentes dos esquemas conceituais são representados **visualmente**, dispensando a necessidade de um prévio conhecimento deste, e podem também ser representado em **módulos**, permitindo uma maior compreensão dos mesmos.
- Mecanismo de interação básico: **manipulação direta** [Shneiderman83].
- Permite a reutilização de consultas na formulação de novas consultas.
- Os esquemas conceituais podem ser constituídos de entidades com aspectos **temporais** e manipulados por **operadores temporais** contidos no sistema.
- A sua interface é simples, possibilitando uma facilidade de aprendizado e utilização, sendo, portanto, adequada a usuários com habilidade técnica limitada e que geralmente ignoram a estrutura interna de um BD.

1.6. Arquitetura Lógica de ConTOM

A figura 1.1 mostra a arquitetura lógica do sistema ConTOM:

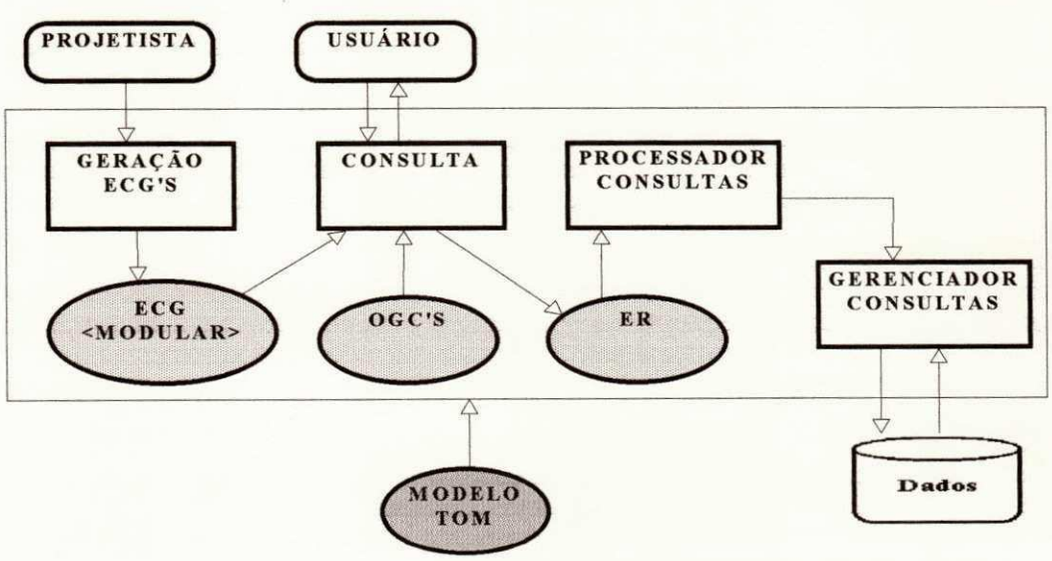


Figura 1.1. Arquitetura Lógica do Sistema Gráfico ConTOM

O projetista é responsável pela geração dos **ECG's** que são os instrumentos básicos para a formulação das consultas feitas pelo usuário (os ECG's gerados podem possuir uma estrutura modular). Novos sub-esquemas dos ECG's gerados pelo projetista, os **ER's**, são gerados pelo usuário através de consultas. O processo de consulta é realizado pelo usuário através da aplicação dos **OGC's** sobre os ECG's. A partir do ER gerado, o **Processador de Consultas** analisa o mesmo, convertendo-o para uma representação interna em que o **Gerenciador de Objetos** possa entender. O **Gerenciador de Objetos** processa a representação interna do resultado da consulta, convertendo-o a um ambiente que gerencia as informações associadas aos objetos componentes dos ECG's e retorna os resultados ao ambiente de consultas. O modelo conceitual utilizado no sistema é o modelo **TOM**, como já foi citado.

1.7. Estrutura da Dissertação

O trabalho é dividido em 3 partes:

A primeira parte, composta pelos capítulos 2 e 3, descreve a abordagem conceitual de aplicação. O capítulo 2 apresenta uma descrição informal sobre modelos de dados semânticos e orientados a objetos. O capítulo 3 define o modelo de dados TOM e a representação gráfica do esquema conceitual sob o modelo TOM, o Esquema Conceitual Gráfico (ECG). Adicionalmente, descreve um ECG dentro de uma abordagem modular.

A segunda parte é composta pelos capítulos 4, 5 e 6, referindo-se ao sistema ConTOM propriamente dito. O capítulo 4 descreve todos os operadores gráficos que podem ser utilizados em uma especificação de consulta sobre um ECG e o formalismo dos mesmos. O capítulo 5 apresenta um estudo comparativo entre algumas interfaces gráficas existentes (incluindo a interface gráfica de ConTOM). O capítulo 6 apresenta o projeto de interface da ferramenta ConTOM.

A última parte, composta do capítulo 7, refere-se às considerações finais do trabalho desenvolvido.

Capítulo 2

Os Modelos "Baseados em Objetos"

2.1. Introdução

Em [Poncelet93], é definida uma classificação que permite considerar duas correntes de abordagens conceituais de uma aplicação: a dos modelos orientados a valor, e a dos modelos semânticos e orientados a objeto.

Como exemplo da primeira corrente temos o modelo **relacional** [Codd70], onde a informação semântica, ligada a uma estrutura complexa de um objeto, se encontra distribuída sob forma de valores entre diferentes relações [Kent79], [Hull89]. Todo objeto, inicialmente estruturado dentro do universo real, é representado por conjuntos de tuplas. A manipulação das entidades é feita através de seus *valores*, ou seja, a "recomposição" de uma entidade necessita de uma ou várias operações de junção. Desta forma, o modelo relacional pode ser qualificado de modelo "baseado em valores". No fim dos anos 70, aparecem os primeiros modelos que, apesar de não estenderem o modelo relacional com novos conceitos, oferecem a possibilidade de definir e manipular estruturas hierárquicas mais complexas. Entretanto, em tais modelos, o aspecto de valor das entidades impõe ainda a utilização de junções não naturais para o usuário, e não possibilitam uma modelagem conceitual "natural", devido ao raciocínio sobre a modelagem sempre ser realizado em termos de entidades [Hull89], [Schek90].

Em oposição à primeira corrente, surgiram os modelos de dados **semânticos** e os modelos **orientados a objetos** como alternativas potenciais para modelar aplicações de BD's avançados, qualificados de modelos "baseados em objetos", e que se apóiam sobre uma equivalência direta entre as entidades do mundo real e uma base de dados [Hull89]. Os modelos semânticos são dotados de qualidades poderosas na representação de uma realidade, graças aos mecanismos de abstração que eles propõem e que permitem uma representação simples de informações complexas. Os modelos orientados a objeto permitem uma modelagem conceitual mais abrangente com o encapsulamento de dados e métodos.

Baseado nas características dos modelos "baseados em objetos" e sua superioridade na representação "natural" dos dados, existe o modelo de dados **TOM**, que possui uma abordagem que se enquadra em um duplo contexto, combinando tanto os aspectos estruturais dos modelos semânticos como os aspectos comportamentais dos modelos orientados a objeto, além de características de ser um modelo aberto, com tempo e versionamento. Iremos, neste capítulo, descrever os aspectos principais das abordagens semânticas e de orientação a objeto, pois faz-se necessário para o entendimento do modelo TOM.

2.2. Aspectos Principais dos Modelos Semânticos

Os modelos semânticos são de melhor poder de expressividade do que os modelos tradicionais (hierárquico, rede e relacional), no sentido de representação das entidades do mundo "real" em um nível conceitual, pois possuem mecanismos de abstração que permitem uma especificação incremental das aplicações, ou melhor, permitem modelar entidades complexas através de um objeto complexo (definidos a partir da estrutura de tipos de objetos primitivos) e com relacionamentos hierárquicos entre classes. Seu domínio de aplicação cobre principalmente a concepção de banco de dados.

Os modelos semânticos surgiram primeiramente como ferramenta para projetos de esquemas conceituais (um esquema poderia ser projetado em um modelo semântico em alto nível, e traduzido para os modelos tradicionais), havendo uma analogia entre o objetivo dos modelos semânticos e as motivações iniciais que impulsionaram no desenvolvimento das linguagens de programação de alto nível. O modelo E-R (Entidade-Relacionamento), por exemplo, é um modelo semântico clássico para a concepção de esquemas relacionais [Teorey86].

Nesta seção, apresentaremos os conceitos fundamentais da abordagem semântica. A idéia é apresentar as funcionalidades que são comuns aos diversos modelos semânticos existentes. Para ilustrar nossa proposta, utilizaremos a notação gráfica do modelo GSM (*Generic Semantic Model)*, definido por R. Hull e R. King [Hull87], que contém as principais funcionalidades de modelos semânticos.

2.2.1. Os tipos Atômicos

A maioria dos modelos semânticos propõe uma representação dos tipos de objetos atômicos, isto é, os que não podem ser decompostos. Dentre estes tipos atômicos, se distinguem:

- Os tipos **Abstratos:** representando as entidades do mundo real sem estrutura interna.
- Os tipos **Imprimíveis:** que representam os objetos de tipos pré-definidos (*string*, inteiro, booleano, etc).
- Os tipos **Livres:** representando os elementos obtidos pela ligação É-UM (generalização/especialização).

Figura 2.1. Tipo Abstrato, Imprimível e Livre

Os objetos abstratos podem possuir uma **identidade** constante, permanente e independente de modificações eventuais. Esta noção de identidade é fundamental para os modelos semânticos, já que podem ser considerados modelos "baseados em objetos". A identificação define uma correspondência completa entre os objetos do banco de dados e os objetos do mundo real. Ela permite exprimir, por exemplo, o fato de que uma pessoa tem uma identidade própria, independente de seu nome ou de seu endereço, e que duas pessoas com mesmo nome ou endereço podem ser distintas. Adicionalmente, esta noção é determinante para a formação de objetos compostos, que são "fisicamente" construídos a partir de outros objetos.

2.2.2. Representação por Construção ou por Atributo

Entre as abordagens semânticas, é possível distinguir duas correntes diferentes com relação à ligação entre tipos de objetos:

1. por **Construção,** onde são utilizados construtores de tipos;
2. por **Atributo,** onde as funções representam as ligações entre tipos.

2.2.2.1. Representação por Construção

A seguir definiremos os construtores de agregação e agrupamento, propostos pelas grande maioria dos modelos. Estes construtores também são conhecidos como abstrações, pois permitem decompor uma descrição complexa em vários níveis de abstração, cada qual mostrando só os elementos essenciais a este nível.

A **Agregação** consiste em agrupar diferentes entidades em uma nova entidade de nível superior, ou seja, em construir um objeto a partir de outros objetos da base de dados. Assim, um endereço, por exemplo, é construído pela agregação de um número, uma rua e uma cidade, como ilustra a figura 2.2:

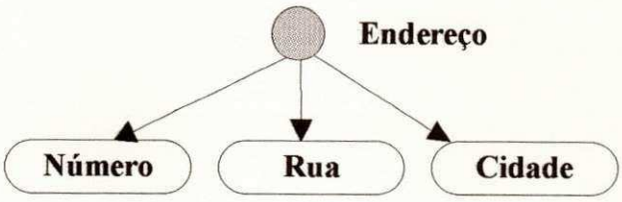


Figura 2.2. Agregação

Formalmente, uma agregação é definida pelo Produto Cartesiano das entidades componentes e sua identidade é determinada pelos valores de seus componentes. Para ser semanticamente válido, a construção deve possuir pelo menos dois componentes.

O **Agrupamento** é utilizado para agrupar um conjunto de objetos do mesmo tipo, por exemplo, um conjunto de pessoas:

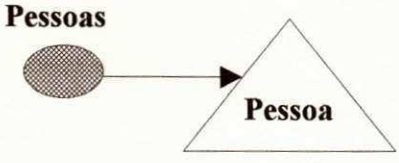


Figura 2.3. Agrupamento

Formalmente, o agrupamento é um conjunto finito de objetos e sua identidade é completamente determinada por este conjunto. Este construtor possui somente um argumento.

As duas construções podem ser aplicadas recursivamente. A figura 2.4 ilustra um exemplo:

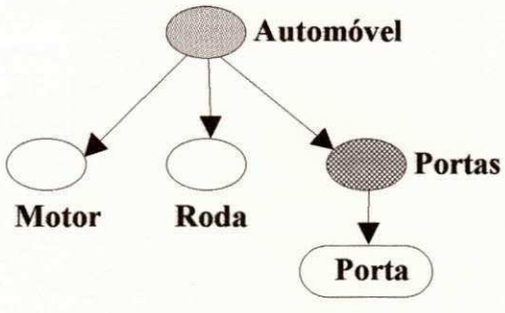


Figura 2.4. Aplicação Recursiva à Agregação e Agrupamento

2.2.2.2. Representação por Atributo

O **atributo** ou **relacionamento** é um dos principais meios a ser utilizado nos modelos semânticos para ligação entre os tipos. Intuitivamente, ele representa uma característica de uma classe de entidades.

O atributo é definido geralmente como uma relação binária dirigida entre dois tipos. Ele pode ser monovalorado (onde será representado como uma flecha), que associa um elemento de um tipo a um elemento de outro tipo, ou multivalorado (onde será representado como uma flecha de duas pontas), que associa um elemento de um tipo a vários elementos de outro tipo. Pode-se representar ligações inversas. A figura 2.5. ilustra um exemplo de representação de atributos:

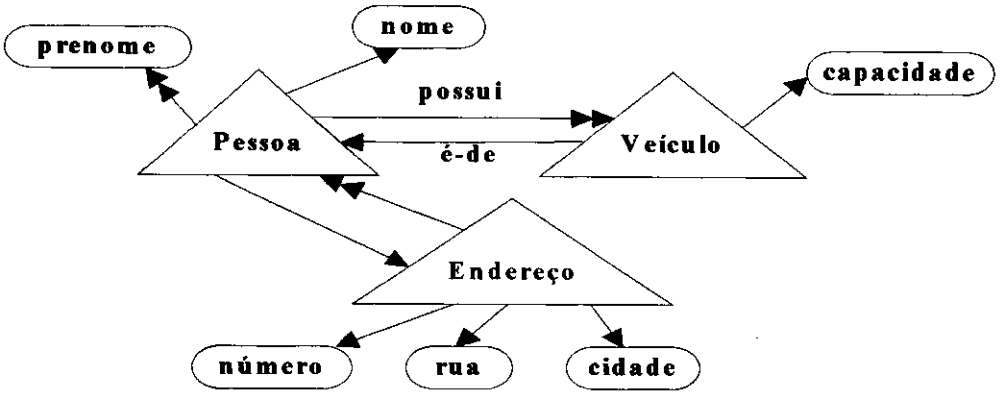


Figura 2.5. Atributos

Em alguns modelos, é feita uma distinção entre atributo e relacionamento como, por exemplo, no modelo E-R [Chen76].

2.2.2.3. Consequências sobre as duas representações

Para ilustrar a dualidade de representação, nós utilizamos um exemplo na figura 2.6, inspirado em [Hull87], representando o seguinte fato: "A uma pessoa e a um veículo corresponde um número de seguro". Este exemplo indica as diferentes modelagens obtidas, utilizando, seja uma abordagem mista (a), seja uma abordagem por construção (b), seja uma abordagem por atributo (c,d).

O exemplo (a), descreve um número de seguro associado à agregação entre "Pessoa" e "Veículo". A cada par de valores, então, é associado este número. O exemplo (b) descreve como uma associação ternária necessita de uma restrição de chave, para garantir que o número de seguro é ligado a um par (Pessoa, Veículo). O exemplo (c), mostra um tipo ligado aos valores dos atributos. É necessário definir uma restrição para especificar o par "principal". Enfim, o exemplo (d) utiliza simplesmente uma função com dois argumentos.

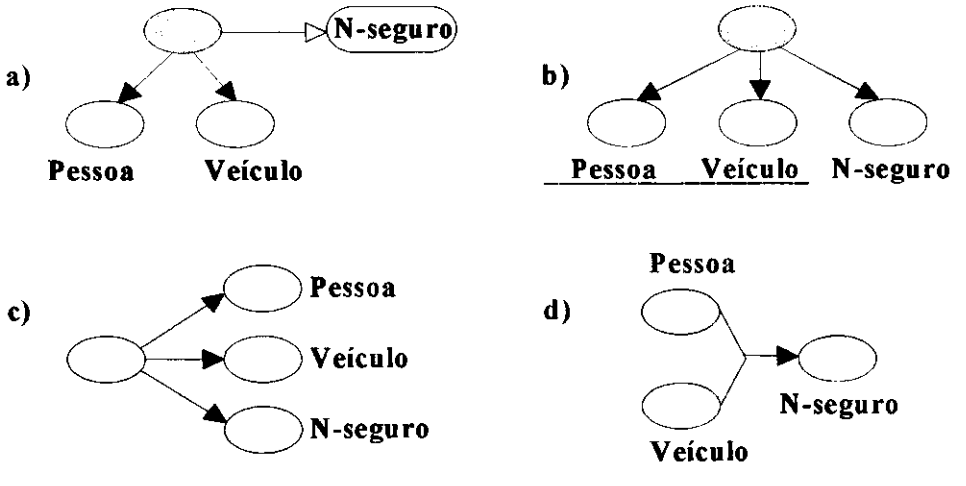


Figura 2.6. Representações diferentes

É uma questão delicada decidir qual a melhor representação para especificar uma aplicação. Entretanto, R. Hull julga a representação por atributo mais conceitual que a representação por construção. Porém, utilizar somente esta representação impossibilita a modelagem de uma relação hierárquica ou recursiva [Schek90]. Alguns modelos propõem uma só representação, outros propõem os dois tipos.

2.2.3. A Ligação É-UM

O terceiro componente essencial dos modelos semânticos é a ligação **É-UM**. Intuitivamente, a ligação É-UM de um **SUBtipo** para um **SUPERtipo,** indica que cada objeto do SUBtipo é elemento do SUPERtipo. Os atributos do SUPERtipo são herdados automaticamente pelo SUBtipo. Desta forma, as ligações É-UM formam um grafo dirigido denominado "hierarquia de herança". As ligações É-UM podem ser vistas de duas formas distintas:

1. Por **Especialização,** que consiste na definição de subconjuntos correspondentes aos diferentes papéis que uma entidade é, eventualmente, obrigada a desempenhar no decorrer de sua vida. A entidade pode mudar de papél sem modificar suas características iniciais ou sua identidade, devido à dependência ao superconjunto. Por exemplo, consideremos o SUPERtipo *Pessoa* e o SUBtipo *Empregado*, um empregado pode parar de trabalhar, porém, ele não deixa de ser uma pessoa. A utilização deste conceito de subtipo é uma ótima maneira de reutilização de definições existentes. Por exemplo, na figura abaixo, o tipo *Empregado* possui todas as características definidas pelo seu supertipo *Pessoa*, como um nome, um endereço, etc. Também é possível especificar novas características aos subtipos, por exemplo, aos empregados é fornecida uma porcentagem de redução sobre os reparos dos veículos, ilustrado na figura 2.7:

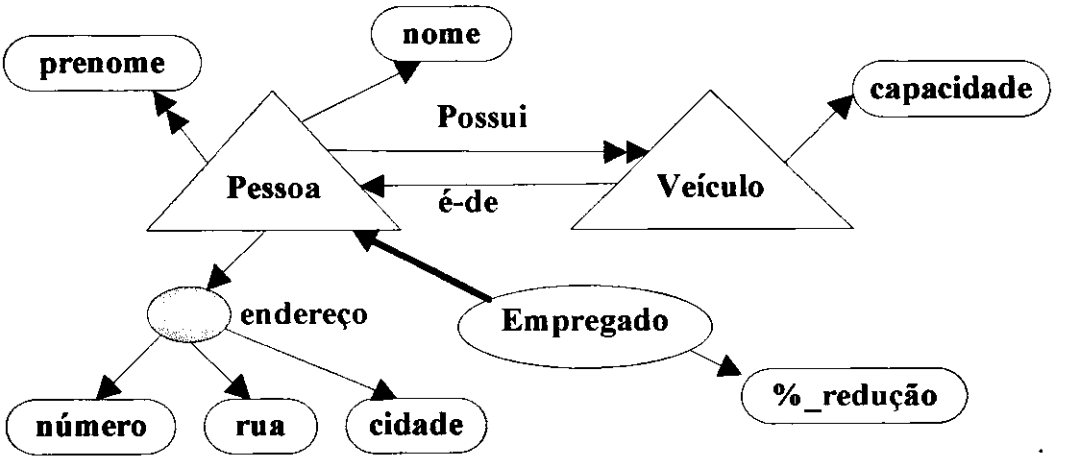


Figura 2.7. Subtipo "Empregado"

2. Por **generalização,** que corresponde à união de tipos já existentes, ou seja, consiste na formação de um tipo virtual como a possível combinação dos diferentes tipos participantes. Por exemplo, na figura 2.8, o tipo *Automóvel* e o tipo *Caminhão* são combinados para formar o tipo *Veículo*. Este princípio é acompanhado geralmente de uma restrição especificando que o supertipo é "coberto" pelos seus subtipos, ou seja, no exemplo, o conjunto de instâncias de *Veículo* é igual ao conjunto de instâncias de *Automóvel* e *Caminhão*.

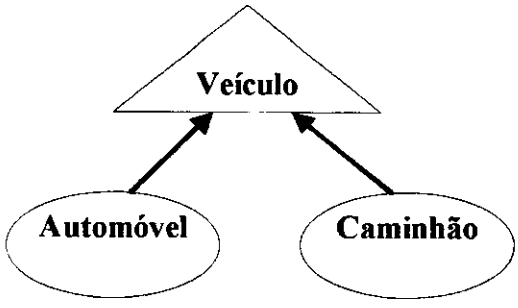


Figura 2.8. Supertipo "Veículo"

É importante ressaltar que a herança dos atributos, resultante da noção de subtipo, é bastante diferente da herança de métodos dentro das linguagens de programação ou os bancos de dados orientados a objeto [Hull87]. Um atributo representa o fato que um valor de propriedade pode ser diferente para cada instância de um tipo, enquanto que um método é comum a todas as instâncias deste tipo, ou seja, o método é compartilhado pelo conjunto das instâncias do tipo. Em resumo, a herança de atributo indica que as características estruturais são transmitidas enquanto que a herança de método significa compartilhamento de código.

2.2.4. Outras Particularidades

Nesta seção, apresentamos brevemente certas particularidades propostas pelos diferentes modelos semânticos.

2.2.4.1. As Restrições de Integridade

Os componentes estruturais dos modelos semânticos fornecem meios de expressão muito poderosos, entretanto existem restrições que são impossíveis de representar utilizando somente os mecanismos propostos. As **restrições de integridade** visam justamente a atender o poder expressivo destes modelos. Numerosas restrições foram expressas nas abordagem semântica e as duas seguintes são as mais utilizadas:

1. As restrições de cardinalidade que especificam as associações (1:1, 1:n, n:1 e n:m) entre as entidades.

2. As restrições que indicam se um atributo é parcial ou total, isto é, que seus valores são respectivamente facultativos ou obrigatórios. Por exemplo, o prenome de uma pessoa pode ser considerado como um atributo parcial enquanto que seu nome é um atributo total.

A figura 2.9 apresenta certas restrições dentro do exemplo proposto pela figura 2.5:

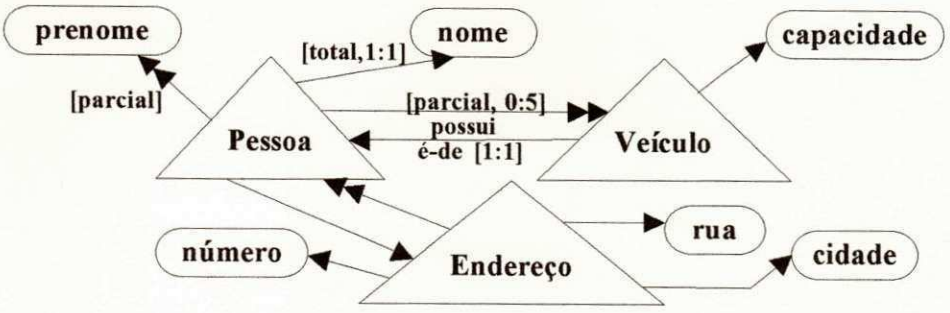


Figura 2.9. Restrições associadas ao esquema

2.2.4.2. Aspectos Temporais

Informações temporais, tais como valores temporais, restrições temporais e características de evolução temporal, estão presentes em um grande número de aplicações do mundo real. A noção de tempo, como datas, períodos, duração de validade de informações, intervalos temporais, surge em diferentes níveis: a) na modelagem de dados, b) na linguagem de recuperação e manipulação dos dados e c) no nível de implementação do SGBD. A modelagem de aspectos temporais é um importante tópico dentro dos modelos de dados. A possibilidade de armazenar, manipular e recuperar dados temporais deve ser considerada quando da escolha de um método de modelagem .

Diversos modelos de dados foram propostos nos últimos tempos, destacamos os modelos: **THM** [Schiel83], **HDRM** [Clifford87], **TDM** [Segev87], **TEER** [Elmasri90], **ERT** [Loucopoulos91], **IXRM** [Lorentzos93], **TRM** [Navathe93], etc, traduzindo a preocupação da comunidade científica com a representação do tempo na modelagem. A grande maioria dos modelos é constituída por extensões temporais realizadas sobre modelos de dados já existentes para cobrir também os aspectos temporais. Novos modelos são também propostos, envolvendo todos os paradigmas de modelagem existentes.

Basicamente, os modelos de dados apresentam as seguintes características na modelagem do tempo:

- Permitem a definição de dados variando no tempo.
- Incluem informações temporais para entidades, relacionamentos, superclasses, subclasses e atributos.
- Utilizam o intervalo como unidade temporal básica, podendo a granularidade temporal (mês, dia, hora, minuto, segundo, etc) ser ajustada à aplicação.
- Cada entidade ou instância de um relacionamento é associada com um elemento temporal representando a duração de vida dos mesmos. O elemento temporal pode ser uma união finita de intervalos temporais, uma classe com períodos de tempo.
- Permitem a evolução temporal de esquemas.
- Definem eventos temporais.

2.3. Aspectos Principais dos Modelos Orientados a Objetos

Os modelos de dados orientados a objetos surgiram a partir dos modelos semânticos de dados, descritos na seção anterior. Em [Bertino92], os modelos semânticos são classificados como modelos **estruturalmente** orientados a objetos, pois não incluem o encapsulamento dos métodos aplicados aos objetos. Os modelos de dados orientados a objetos, portanto, herdam características dos modelos semânticos, acrescidos a conceitos de orientação a objeto surgidos das linguagens de programação, tais como: identidade de objetos, classes, métodos e mensagens, encapsulamento, herança, polimorfismo e *late binding*. A busca de uma padronização de um modelo orientado a objeto tem calcado esforços de pesquisadores. Critérios de padronização podem ser vistos em [Bancilhon89].

A principal contribuição dos modelos de dados orientados a objetos foi suprir as deficiências dos modelos tradicionais que se tornaram inadequados em manipular aplicações cada vez mais avançadas, envolvendo: inteligência artificial, hipermídia/multimídia, CAD/CAM, automação de escritório, interfaces gráficas, entre outras. Em tais aplicações, torna-se necessário a manipulação, o gerenciamento, a manutenção de integridade e a segurança de componentes complexos.

Nesta seção, apresentaremos os conceitos fundamentais de orientação a objeto que são utilizados em banco de dados e os aspectos peculiares de banco de dados tradicionais, dentro de um contexto de orientação a objeto.

2.3.1. Conceitos de Orientação a Objetos para Banco de Dados

2.3.1.1. Objeto e Identidade de um Objeto

Um **objeto** modela os dados que representam uma entidade do mundo real, consistindo de um **estado**, definido por um conjunto de valores específicos de sua estrutura interna, ou seja, valores dos seus atributos, e um **comportamento**, definido por um conjunto de métodos, que representam as operações que atuam sobre o seu estado.

Em linguagens de programação orientadas a objeto, os objetos são entidades computacionais ativas em vez de dados passivos nas linguagens convencionais. Um objeto é composto por uma parte visível, sua **interface,** que corresponde às especificações das operações que fazem parte do seu comportamento, e uma parte "escondida", sua **implementação,** onde compreende os dados, sua estutura e o código das operações. Adicionalmente, todo objeto possui uma identidade única (ver seção 2.1.1).

Em resumo, podemos definir um objeto da seguinte maneira:

OBJETO = Estado (dados) + Interface (especificação) + Implementação (código)

2.3.1.2. Métodos e Mensagens

Como já foi observado, as operações aplicadas aos objetos são chamadas de **métodos** e são o único meio de alterar o estado do objeto. Objetos podem ser ativados através do **envio de mensagens.** Um envio de mensagem corresponde a aplicação de um método a um objeto e possui a seguinte estrutura: o identificador do objeto **receptor**, o **seletor** da operação (nome do método) sobre o objeto e os **argumentos** (opcionais) pertencentes à operação (ressaltando novamente que as operações devem fazer parte da interface do objeto receptor). A execução de um método em um dado objeto geralmente envolve envio de mensagens para outros objetos, a fim de executar outros métodos, e assim por diante.

As mensagens, apesar de serem similares, não correspondem à chamadas a procedimentos em um ambiente convencional. As operações são associadas somente ao objeto, que contém o código. É permitido, portanto, haver vários métodos de mesmo nome, definidos em vários objetos diferentes, onde cada método tem um comportamento diferente dos demais. Considere, por exemplo, um método denominado *imprimir*, que se comporta de maneira particular, conforme a natureza do objeto, diferentes métodos com o mesmo nome *imprimir* podem ser definidos. Os códigos destes métodos, então, são associados a cada objeto correspondente. Em um ambiente convencional, não é possível associar um mesmo nome de método para vários tipos de implementações, deve-se utilizar nomes diferentes. Em orientação a objeto, existirão tantas implementações do método *imprimir* quantas forem necessárias, além do que o programador não será obrigado a testar tipos de objetos dentro do seu programa.

Esta característica, conhecida como **polimorfismo,** é uma das diferenças fundamentais entre as linguagens tradicionais e as linguagens orientadas a objeto. Durante o envio de mensagem, o corpo do método a ser executado será associado ao objeto receptor em tempo de execução. Este processo de seleção do método na execução se chama ***late binding.*** *Late binding* oferece um grande dinamismo ao sistema, pois um mesmo envio de mensagem produz resultados diferentes conforme o tipo de objeto para o qual se aplica. Entretanto, este mecanismo pode comprometer a performance do sistema.

2.3.1.3. Encapsulamento

Como já foi dito, um objeto tem uma interface que constitui a parte visível do objeto, e uma implementação, que constitui a parte "escondida" para o usuário do objeto, o qual não pode manipular o objeto senão através das operações especificadas na sua interface. Este é o princípio de **encapsulamento**.

A aplicação de encapsulamento em banco de dados é um pouco diferente das linguagens de programação. Nas linguagens, somente a especificação das operações faz parte da interface. Em BD's, onde as necessidades diferem um pouco das linguagens de programação, tudo é centrado sobre os dados, desta forma, é interessante que a estrutura dos dados faça parte da interface, ao invés da implementação, pois o usuário possui necessidade de acessar essa estrutura (em alguns sistemas de BD's, a estrutura é parte integrante da interface), mesmo que não possa ser diretamente alterada.

Do ponto de vista de BD's, o interesse de encapsulamento é múltiplo. De um lado, ele garante uma boa modularidade dos programas, colocando o objeto como uma unidade de modularização, por outro lado, ele garante a independência lógica dos dados. A implementação de um objeto pode ser modificada sem que a utilização deste objeto seja afetada.

2.3.1.4. Classes e Tipos

A noção de **classe** permite agrupar objetos que possuem características comuns, a fim de facilitar seus gerenciamentos. Uma classe tem um nome, uma descrição da estrutura dos objetos e métodos. Os objetos são denominados as **instâncias** da classe. O conjunto das instâncias de uma classe formam sua **extensão**. A noção de classe já foi introduzida na seção anterior (seção 2.2), onde ela é utilizada como noção de **tipo** em modelos semânticos para agrupar as entidades de mesma natureza. Entretanto, como já foi observado, os modelos orientados a objeto são mais ricos, visto que eles não descrevem somente a estrutura dos objetos como também os métodos que se aplicam aos mesmos. Em outras palavras, as classes implementam **tipos de dados abstratos** e diferem dos mesmos em relação a checagem de tipo, isto é, o tipo de dado abstrato permite a checagem de tipo em tempo de compilação enquanto que a classe permite a checagem de tipo em tempo de execução.

2.3.1.5. Herança

A **herança** é um dos conceitos mais importantes de orientação a objeto. Ele consiste em extrair partes comuns de classes distintas e agrupá-las em uma classe de nível mais alto. Esta última, é denominada **superclasse** e as primeiras, são as **subclasses**. As subclasses **herdam** os atributos e métodos da superclasse além de possuírem atributos e métodos próprios. A herança reune os conceitos de especialização e generalização dos modelos semânticos.

Existe uma situação em que uma classe herda propriedades de várias superclasses simultaneamente, tal situação é denominada de **herança múltipla,** onde a classe possui todas as propriedades de suas superclasses. Este conceito pode ocasionar situações de **conflito**, que ocorre quando um mesmo atributo ou

método existe para duas ou mais superclasses. O conflito pode ser resolvido de diversas maneiras, porém, não entraremos em detalhes no nosso trabalho.

A herança é um conceito interessante por vários pontos de vista. É um bom mecanismo de abstração, o qual permite obter uma descrição modular. A herança também permite evitar repetições do código. Enfim, a herança é um dos melhores meios para a reutilização de código.

2.3.1.6. Extensibilidade

Os sistemas orientados a objeto possuem um conjunto de tipos e de classes pré-definidos. Estes tipos ou classes podem ser genéricos ou específicos. Dependendo da necessidade, pode-se estender este conjunto, definindo novos tipos ou classes ou especializando o que já existe. Tais extensões são automaticamente integradas ao sistema, sem afetar os tipos ou classes pré-definidos. Esta característica de orientação a objeto oferece uma grande flexibilidade à programação.

2.3.2. Banco de Dados Orientado a Objeto (BDOO)

Na seção anterior, apresentamos os conceitos gerais de orientação a objeto que são utilizados em banco de dados. Estes conceitos se referem à modelagem (objeto, classe, herança) e à programação (encapsulamento, polimorfismo, extensibilidade,etc). Agora, vamos apresentar aspectos mais específicos a banco de dados, sem a pretensão de dar uma definição completa de sistemas de banco de dados orientados a objeto.

2.3.2.1. Objetos Complexos

Os objetos possuem uma estrutura que é primordial para BD's, onde deseja-se representar infomações complexas. Os modelos orientados a objeto constróem objetos complexos a partir de objetos atômicos, e de construtores que se aplicam independente do tipo dos objetos (ver seção 2.1.1). O modelo que suporta a construção de objetos complexos estruturados, adquire um poder de modelagem uniforme e homogêneo, permitindo que os objetos complexos sejam manipulados por novas aplicações.

2.3.2.2. Identidade do Objeto

A **identidade** do objeto é uma noção essencial a BD's, pois é importante haver distinção de cada objeto dentro da base de dados. Este conceito foi apresentado na seção 2.1.1, e agora será apresentado com mais detalhes.

A fim de garantir uma boa coerência semântica, o **identificador,** que representa a identidade do objeto, deve possuir dois princípios. Primeiro, ele não deve estar ligado a um valor do objeto. Segundo, contrário aos sistemas relacionais, ele não deve ser gerado por um programador ou usuário que não podem controlar todas as ocorrências do objeto, e sim deve ser gerado internamente pelo sistema. O sistema deve garantir a unicidade de seus identificadores, mesmo para objetos que possuam o mesmo estado. O identificador, seja um endereço físico ou lógico, permite acesso ao objeto. Ele se comporta de maneira análoga a um apontador de uma linguagem de programação e é sempre invisível ao usuário ou ao programador.

As linguagens que manipulam objetos com identidade necessitem de operações que permitam tratar deste conceito. Em particular, este conceito define diferentes noções de comparações de objetos. Nós distinguimos comparação de identidade, que consiste em comparar os identificadores à comparação de igualdade, que consiste em comparar estados. Maiores detalhes podem ser vistos em [Khoshafian90].

2.3.2.3. Persistência

Dentro das linguagens de programação, os dados não sobrevivem além da execução de um programa. Em BD's, a **persistência** é fundamental, todos os dados do BD são persistentes. Em geral, a persistência não necessita de comando particular, ela é ligada à noção de tipo.

A persistência deve obedecer aos seguintes princípios:

1. A persistência deve ser independente de tipagem. Todo dado deve poder ser persistente, não importando seu tipo.

2. A persistência deve ser independente do modo de armazenamento.

Existem várias maneiras diferentes de realizar a persistência, entre as quais:

- Ligado ao tipo: decide-se ao nível de esquema quais são os tipos que são persistentes.Todos os dados de tipos persistentes são persistentes.

- Ligado a uma variável: decide-se se uma variável é persistente ou temporária.

- Decidido na criação do objeto: onde existem dois modos de criação, o modo persistente e o modo temporário. A decisão de utilizar um ou outro fica a cargo do usuário.

- Dinâmico: um objeto pode mudar de estado persistente para temporário, ou vice-versa.

Dentro do contexto de BD's, os pontos de entrada correspondem a certos elementos do esquema. Em BDOO's, a definição de um esquema compreende os conceitos de classe e de instâncias nomeadas. Em certos casos, os nomes das classes projetam a extensão da classe e são os pontos de entrada da base, em outros sistemas, o usuário que projeta as raízes da persistência, que podem ser as classes, os objetos ou variáveis.

2.3.2.4. Linguagem de Consultas

Um sistema de BD's deve fornecer uma linguagem que permita exprimir facilmente consultas simples ou complexas. Tal linguagem deve responder a vários critérios e deve ser de alto nível. As linguagens declarativas, onde o usuário descreve o resultado e não o meio de obter o mesmo, satisfazem a este ponto. Ela deve ser otimizável. Há geralmente muitos meios de obter um mesmo resultado. Alguns são mais eficazes que outros, entretanto, o usuário nem sempre tem a possibilidade de saber, pois a eficácia depende de implantação. Portanto, é indispensável que as consultas possam ser otimizadas pelo sistema, antes da sua execução. Enfim, tal linguagem não deve ser ligada a uma aplicação particular, mas deve poder ser aplicada sobre toda a base de dados.

A linguagem não deve conter operações definidas especificamente por uma aplicação do usuário, nem oferecer facilidades de consulta apropriadas a uma aplicação específica. Em sistemas relacionais, este tipo de serviço é fornecido pelo SQL. Nos sistemas orientados a objeto, as consultas podem ser expressas através dos métodos ou pela navegação sobre a as classes componentes de um esquema de BD.

Em modelos relacionais, o resultado das consultas é sempre uma relação. Nos modelos OO, podemos supor que o resultado de uma consulta é um objeto. O problema é saber qual a classe deste objeto. Como a complexidade deste modelo é mais rica que o relacional, não deve importar o tipo do resultado, qualquer tipo, a priori, deve ser aceito pelo modelo.

Um último ponto importante refere-se a forma de utilização das linguagens de consulta, que podem ser dividida em duas: de forma *ad hoc* ou com uma linguagem de programação, sendo nesta última a linguagem de consulta independente ou integrada à linguagem de programação.

Capítulo 3

O Modelo TOM e o ECG

3.1. Introdução

O modelo **TOM** *(Temporal Object Model)* [Schiel91, David92] é um modelo de dados orientado a objetos. A estrutura do modelo herdou as características do modelo semântico THM *(Temporal Hierarchic Data Model)* [Schiel83] e veio adicionada com as características de orientação a objeto. O modelo possui, além das facilidades de um modelo orientado a objetos, regras de integridade associadas a cada método através de pré-condições e pós-condições, controle de eventos e regras, representação de tempo e de versões, e a filosofia de um ambiente aberto orientado a objetos.

Como já foi observado no capítulo 1, a especificação visual de um esquema conceitual sob o modelo TOM, é denominado de **Esquema Conceitual Gráfico - ECG**. Um ECG é a unificação do modelo TOM com uma representação gráfica que contém os aspectos estáticos do modelo. Considerando que o componente mais importante de qualquer interface de BD's é a representação visual dos esquemas conceituais, o ambiente ConTOM tem o ECG como componente **central**, sobre o qual se aplica todo o processo interativo de acesso, manipulação e consulta.

O capítulo é dividido em duas partes: a primeira parte descreve o modelo TOM em maiores detalhes, segundo seus aspectos estáticos e dinâmicos. Nós limitaremos aqui em dar somente uma descrição informal do modelo. A descrição formal ou maiores informações podem ser encontradas em [David92]. A segunda parte descreve o ECG dividido nas seguintes atividades: especificação, representação gráfica, modularização e acesso.

3.2. Aspectos Gerais do Modelo TOM

O modelo TOM parte do princípio de que o universo é dividido em três mundos: o mundo abstrato, o mundo concreto e o mundo do modelo. O universo do discurso é a parte do mundo real (abstrato e concreto) que é relevante para uma aplicação específica. A modelagem se realiza pela representação do mundo real no mundo de modelos, abstraindo-se todos os detalhes desnecessários à aplicação.

O modelo envolve dois aspectos: o **estático**, que se refere às estruturas dos dados em si e é baseado nos conceitos de classe, relacionamento, hierarquias de abstração e tempo, e o **dinâmico**, que se refere-se ao comportamento da aplicação decorrente das mudanças de estado (evolução) dos objetos em consequência da ocorrência de eventos, e é baseado nos conceitos de método, evento, *trigger,* regra e versões.

3.2.1. Aspectos Estáticos

3.2.1.1. Classe

Toda e qualquer entidade do mundo real é representada por um objeto instância de uma **classe**, dentro do modelo. Cada classe de objetos apresenta uma estrutura de relacionamentos e métodos que é herdada por todos os objetos instâncias daquela classe.

De acordo com o tipo de suas instâncias, as classes no modelo são classificadas como: **primitivas,** que correspondem aos objetos imprimíveis nos modelos semânticos e cujas instâncias são identificadas por tipos de dados simples (inteiro, *string,* etc), **não primitivas,** que correspondem aos tipos abstratos nos modelos semânticos e cujas instâncias são identificadas por um ou mais relacionamentos (ex: *empregado)* e **temporais**, cujas instâncias, quando removidas da classe, permanecem no BD com uma indicação do tempo de duração em que foram membros da mesma (ver seção 3.2.1.5).

Em [David92], considera-se uma classe como um sistema composto de duas partes:

1. **A descrição da classe (intenção):** composta do nome da classe, uma lista de relacionamentos, uma lista de métodos, informação sobre a sua posição na hierarquia de classe, suas subclasses e superclasses, e parâmetros de tempo.

2. **O conjunto de instâncias (extensão):** composto de um conjunto de objetos em que os relacionamentos descritos na classe assumem valores concretos para cada objeto (o seu estado), e métodos da classe a que pertencem.

3.2.1.2. Metaclasse

A área de BD's vem sofrendo grande influência dos chamados sistemas abertos. Tal influência se reflete particularmente na concepção de um modelo de dados "aberto" [David92]. O modelo TOM é considerado um modelo aberto pois possui um **metanível** no qual é possível alterar o próprio modelo de dados.

O metanível é composto por várias **metaclasses**, que descrevem a sintaxe (estrutura) e a semântica (comportamento) do modelo de dados e são independentes de aplicações. No metanível, o Administrador de Modelos (AM) pode criar ou modificar um modelo de dados específico, especificando novos conceitos e abstrações inerentes ao mesmo, que serão representados como metaclasses. Em suma, o AM modela o modelo [David92]. Para o TOM, a principal metaclasse pré-definida, é denominada *TOM_Class*. A partir dela, podem ser criadas, como suas instâncias, classes e outras metaclasses. A nível de aplicação, o projetista cria classes de aplicação como instâncias ou subclasses das metaclasses do metanível, definidas pelo AM. Desta forma, ele utiliza as metaclasses para representar conceitos do modelo que são necessários para um sistema específico.

A figura abaixo ilustra os níveis de abstração do modelo TOM:

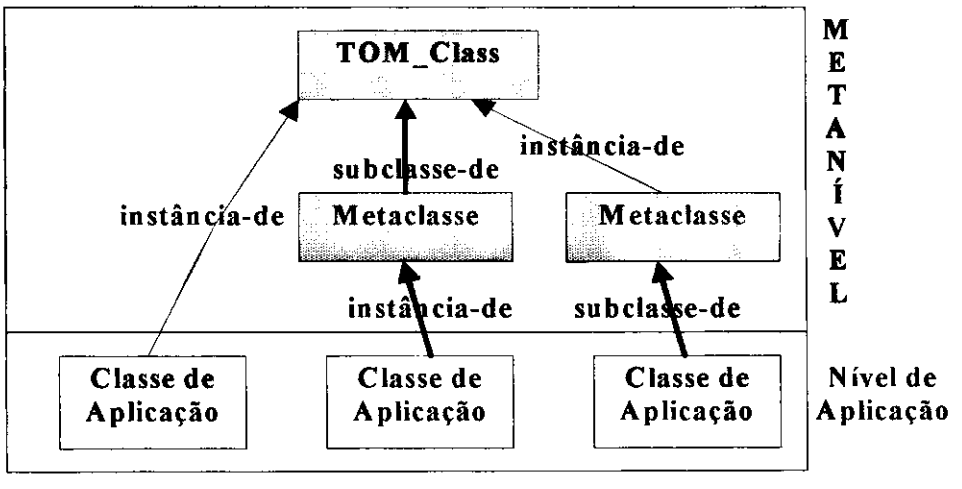


Figura 3.1. Níveis de Abstração do TOM

3.2.1.3. Relacionamento

Um **relacionamento** no modelo TOM representa a informação sobre dois objetos que possuem uma associação válida e corresponde à representação por atributos em alguns modelos semânticos. O TOM trata relacionamentos como unidades manipuláveis, diferente de outros modelos de dados orientados a objetos, que tratam o relacionamento entre os objetos através das variáveis de instância (atributos) que referenciam outros objetos.

Em [Rumbaugh87], é analisado esta questão entre modelos orientados a objeto que conseguem expressar classes e relacionamentos hierárquicos entre classes, porém, não existe um suporte sintático ou semântico para expressar o relacionamento diretamente, isto é, um relacionamento não é tratado como um elemento que possa ser manipulado como uma unidade, a informação sobre um relacionamento é distribuída entre diferentes classes. A representação de relacionamento entre objetos, através de uma variável de instância em cada objeto, falha ao capturar a semântica de um relacionamento, além de expor muito da implementação por causa da escolha das variáveis de instância e métodos, não permitindo, desta forma, a não distinção entre os níveis conceitual e de implementação na hora de representar relacionamentos.

É importante, portanto, descrever um sistema através de um modelo de dados orientado a objetos, representando não somentes as classes, mas os relacionamentos entre as mesmas, pois abstrai a estrutura de alto nível do sistema, sem haver necessidade desta estrutura ser dependente da especificação da implementação particular das classes e seus métodos. O modelo TOM satisfaz plenamente este requisito.

Cada relacionamento no TOM possui uma cardinalidade, como uma restrição de integridade, expressa pelo par de valores (*min*, *max*) que determina o mínimo e o máximo de instâncias associadas pelo relacionamento, ou seja, cada instância de uma classe está associado a no mínimo *min* e no máximo *max* instâncias de outra classe. A cardinalidade é um importante aspecto do mundo real sendo abstraído, e está ausente em muitos modelos.

Os seguintes tipos de relacionamentos estão presentes no modelo:

1. **Relacionamento-Instância:** é aquele que relaciona instâncias de uma classe com instâncias de outra classe. Por exemplo, o relacionamento *tem-nome* relaciona instâncias da classe *empregado* com instâncias da classe *nome*.

2. **Relacionamento-Classe:** é aquele que relaciona a classe como um todo a uma instância de outra classe, isto é, não varia de instância para instância. Os relacionamentos-classe servem para considerar a classe como um objeto com seus próprios relacionamentos e para fatorar valores comuns a todas as instâncias. Por exemplo, o relacionamento *consumo-máximo* relaciona a classe *Fabricante* como um todo, a uma instância da classe *consumo-combustível*, enquanto que o relacionamento *número-rodas* expressa uma propriedade comum a todas as instâncias da classe *Automóvel*.

3. **Relacionamento Temporal:** representa o relacionamento pré-pós e o relacionamento com valores anteriores, a serem definidos na seção 3.2.1.5.

3.2.1.4. Abstrações Hierárquicas

O modelo TOM suporta todos os mecanismos de abstração dos modelos semânticos. Estas abstracões são às vezes chamadas de relacionamentos classe-a-classe, pois não fazem referência a instâncias específicas, mas expressam propriedades comuns a todas as instâncias das classes relacionadas. As abstrações podem ser aplicadas recursivamente na construção de objetos complexos e são descritas abaixo.

3.2.1.4.1. Generalização

É representada pelo predicado ***é-um*(A, B)** significando que A é uma **subclasse** de B (ver capítulo 2, seção 2.2.3). O inverso é chamado de **especialização**, aplicando um **papel** ou critério de especialização a uma classe. O papel da especialização dos objetos pode ser dado por um predicado, uma enumeração ou índice. Por exemplo, podemos ter uma classe *pessoa,* superclasse das classes *empregado, desempregado* e *estudante*, segundo o critério de especialização *cond_trabalho.*

3.2.1.4.2. Agregação

É representada pelo predicado ***é-parte*(A,B)** onde A é um **componente** da classe agregada B (ver capítulo 2, seção 2.2.2.1). Por exemplo, cada objeto da classe *corpo-humano* é constituído de objetos das classes *cabeça, tronco e membros.*

3.2.1.4.3. Agrupamento

É representada pelo predicado ***é-elemento*(A,B)** onde as instâncias de B são **conjuntos de instâncias** de A (ver capítulo 2, seção 2.2.2.1). B é chamado de classe-grupo e A é a classe elemento. Por exemplo, grupos de objetos da classe *árvore* formam instâncias da classe *floresta.*

3.2.1.4.4. Herança nas Hierarquias

Na generalização, a herança é **automática** pois representa o relacionamento *é-um.* Na agregação e agrupamento, define-se uma herança **seletiva** para cada relacionamento e método, identificada em:

- Herança **Direta**: onde relacionamentos e métodos podem ser herdados pelos objetos de nível inferior, sem modificações. Por exemplo, a classe *automóvel* possui o relacionamento *tem-dono* e o método *muda-dono* que podem ser herdados pelas classes componentes *motor, porta e chassis*.

- Herança **Computada**: onde algumas propriedades não devem ser herdadas diretamente, mas são obtidas através de uma computação. Por exemplo, o método *inclui_casal* pode ser a junção dos métodos *inclui_marido* e *inclui_esposa*, ou o relacionamento *peso* é a soma dos pesos dos componentes.

Contudo, existem relacionamentos e métodos que só se aplicam aos objetos compostos. Por exemplo, o relacionamento *tem-placa* se aplica somente à classe *automóvel* e não aos seus componentes.

3.2.1.5. Aspectos Temporais

Um dos principais fundamentos do modelo TOM é a modelagem do tempo. O tempo é modelado através das classes temporais, dos relacionamentos temporais e pré-pós.

3.2.1.5.1. Classes Temporais

Em uma classe temporal, a cada instância está associado o tempo de duração histórico do objeto naquela classe. Este tempo é dado por um (ou vários) intervalos temporais. Para objetos atuais, o limite superior deste intervalo deve ser uma data no futuro ou é uma variável especial, representando o presente.

3.2.1.5.2. Relacionamento Pré-Pós

Há situações em que um objeto, ao ser eliminado de uma classe, deve passar a pertencer a uma outra classe de objetos. De modo inverso, ocorrem casos onde um objeto incluído em uma classe deve vir de uma classe de objetos imediatamente anterior.

Esse relacionamento mútuo entre duas classes é denominado, no TOM, de relação **pré-pós**, denotado por c1 >-----> c2, onde c1 é a **pré-classe** e c2 a **pós-classe**. Existem variações neste tipo de relacionamento:

- Se todos os objetos que saem de c1 vão para c2, o relacionamento é dito **pré-exclusivo/pós-simples**, denotado por c1 >>-----> c2.

- Se todos os objetos que estão em c2 vieram de c1, então o relacionamento é dito **pré-simples/pós-exclusivo**, denotado por c1 >----->> c2.

- O relacionamento pode ainda ser **pré-exclusivo/pós-exclusivo**, significando que todos os objetos que saem de c1 vão para c2 e todos objetos que estão em c2 vieram de c1, denotado por c1 >>----->> c2.

Os relacionamentos pré-pós permitem estabelecer históricos da evolução de um objeto. Pode-se considerar, como exemplo, uma biblioteca cujos exemplares podem estar nos seguintes estados: disponível, reservado, emprestado e perdido. Estes estados são representados como classes. A classe *disponível,* por exemplo, é pré-exclusivo e pós-simples da classe *emprestado* e pré-simples da classe *perdido.*

3.2.1.5.3. Relacionamentos Temporais

Considera a característica temporal nos relacionamentos. Os valores dos relacionamentos alterados serão mantidos no banco de dados para efeito de histórico, com a indicação das datas de início e término da existência dos valores anteriores do relacionamento. Como exemplo, pode-se considerar o relacionamento *trabalha-em* entre as classes *empregado e departamento* com o parâmetro "com 10 valores anteriores", indicando o histórico dos 10 últimos departamentos em que o empregado esteve alocado.

3.2.2. Aspectos Dinâmicos

3.2.2.1. Método

Um **método** é um procedimento de mudança de estado dos objetos, que se reflete na alteração dos seus relacionamentos e envio de mensagens a outros objetos. Os métodos são acionados por mensagens que solicitam sua execução.

São os seguintes tipos de métodos presentes no modelo:

1. **Método-Instância:** se a mensagem for enviada a uma instância de uma classe, cujo estado será modificado ou consultado.

2. **Método-Classe:** se a mensagem for enviada a uma classe. Os métodos-classe são usados para manipular relacionamentos-classe, criar novas instâncias, ou selecionar várias instâncias de uma classe.

A nível conceitual, um método em TOM é especificado por um par de pré e pós-condições, que caracterizam a mudança de estado provocada pelo método

3.2.2.2. Evento

Um **evento** é um sinal ou condição que, ao ocorrer, pode vir a mudar o estado de algum objeto. Nos sistema de regras dinâmicas do TOM [Carvalho93], eventos foram definidos como objetos pertencentes à metaclasse *Evento* e às suas subclasses. Os objetos da classe *Evento* possuem: *nome*, a situação que descreve quando ele ocorrerá e o atributo *ativo*, que determina se o evento está ativo ou não.

No modelo TOM, existem três tipos de eventos, cada um descrito em uma subclasse de *Evento:*

1. Evento **Externo**: é aquele que ocorre com a chegada de uma mensagem do mundo real (são as chamadas a métodos, pelo usuário). Possui os atributos *Método* e *Classe* que contém método. Juntos, esses atributos especificam que a execução de um método de uma certa classe configura o evento.

2. Evento **Temporal:** é aquele que ocorre em um dado instante do tempo. Ele é acrescido dos atributos *Quando, Atraso e Tipo.* O atributo *Quando* especifica o ponto inicial do intervalo de ocorrência do evento, o atributo *Atraso* especifica a duração de validade do evento e o atributo *Tipo* se refere ao tipo de evento ser discreto ou contínuo.

3. Evento **Interno:** existem duas formas de ocorrência de eventos internos. Uns ocorrem quando um método do banco de dados é executado (outro evento), outros representam um certo estado do banco de dados (ex: somatória dos salários > dinheiro disponível). Os primeiros são chamados eventos **exógenos** e os outros, eventos **endógenos.**

3.2.2.3. *Trigger*

Trigger é a forma de execução de uma ou mais ações, em consequência da ocorrência de eventos, sob certas condições pré-estabelecidas. Ele pode também executar atualizações para manter a consistência do BD, ou impedir a execução de uma operação de um evento interno exógeno.

São os seguintes atributos de um *trigger*:

- **Ativo:** especifica se o *trigger* está ativo ou não.

- **Prioridade:** especifica a prioridade de execução do *trigger*. Quanto maior a prioridade, mais favorecido será o *trigger* em relação a outros *triggers*.
- **Desabilitado:** atributo que consiste em uma coleção de identificadores dos objetos do banco de dados para os quais o *trigger* não deve disparar. Funciona como uma espécie de tratamento de exceção.
- **Sequência de Execução:** possui 3 valores possíveis: antes, depois ou igual, onde determina se a ação do *trigger* será executada antes, depois ou concomitante à finalização do evento.
- **Condição:** deve ser verdadeira para permitir o disparo da ação do *trigger*.
- **Ação:** é o conjunto de operações a serem executadas, caso a condição do *trigger* seja satisfeita.
- **Falha-Ação:** é o conjunto de operações a serem processadas, caso a condição do *trigger* não seja satisfeita.

3.2.2.4. Regra

Regra é uma associação entre eventos e *triggers*, determinando quais eventos irão ativar quais *triggers*. Após a definição das regras, o atributo *regra* nas informações do evento deve ser preenchido.

3.2.2.5. Versões

Versões são diversas ocorrências de um mesmo objeto que coexistem numa base de dados e representam diferentes estados do mesmo objeto semântico. O conceito de versão no TOM apresenta as seguintes características [Schiel91]:

- Todo objeto versionado possui uma parte genérica (g-objeto) e partes versionadas (v-objetos).
- Referências podem ser feitas ao objeto como um todo ou a versões individuais.
- Assim como os objetos, as versões possuem um identificador único gerado pelo sistema.
- As versões se aplicam a todas as formas de abstração, podendo serem herdadas também.
- Todas as versões de objeto genérico estão relacionadas entre si e formam um grafo de versão, as formas de versionamento são: *Seguinte* (sequência cronológica única de uma versão para a outra), *Alternativa* (as versões seguintes são mutuamente exclusivas), *Variante de* (todos os objetos seguintes são variantes válidas do mesmo objeto) e *Parte de* (decomposição de um objeto em partes que são desenvolvidas independentemente uma das outras).

3.3. Esquema Conceitual Gráfico (ECG)

Nesta seção, descreveremos o ECG dentro das seguintes abordagens: **especificação** de um ECG, que corresponde à etapa de projeto de um esquema conceitual. Esta etapa se baseia na metodologia *POKER* ou na metodologia FADO [Furtado93a], através das quais pode ser gerado um ECG (as duas metodologias serão vistas no decorrer deste capítulo): a **representação gráfica** de um ECG, que corresponde aos componentes gráficos associados aos conceitos de classe, relacionamentos e abstrações e como tais componentes são visualizados; a **modularização** de um ECG, onde tanto o projeto, quanto a representação de um ECG, são analisados dentro de uma perspectiva modular e enfim, o **acesso** a um ECG, onde serão mostrados dois níveis de acesso a um ECG modular ou não.

3.3.1. Especificação de um ECG

Nesta seção, descreveremos brevemente as metodologias FADO e *POKER*, apropriadas para a geração de um esquema conceitual gráfico. Discutiremos aqui apenas a utilização destas metodologias no processo de identificação dos componentes **estruturais** de um esquema conceitual.

3.3.1.1.Metodologia FADO

A metodologia **FADO** [Furtado93a] é uma metodologia orientada a objetos que auxilia no projeto de BD's temporais orientados a objetos, modelando o domínio de aplicação em objetos e eventos. Ela possui documentação gráfica, composta de diagramas, e textual, composta de tabelas e formulários. Entre os diagramas existentes na metodologia, há o diagrama **estático**, diagrama estrutural que projeta as estruturas das classes e relacionamentos entre elas. O ECG corresponde a um diagrama estático na representação FADO. A metodologia FADO é basicamente composta de seis fases descritas abaixo. Cada fase é constituída por uma série de passos que possuem uma numeração correspondente a figura 3.2:

1. **Análise do domínio da aplicação:** é centrada na análise de todas as aplicações de um dado domínio e na identificação dos objetos que fazem parte destas aplicações. Nesta fase, são identificados as classes e os objetos (passo 1), obtendo uma lista de classes e um esboço do diagrama estático. Os outros passos (2 e 3) referem-se a identificação dos estados de objetos e os eventos.

2. **Identificação da semântica das classes:** refere-se à tomadas de decisão do projeto e a obtenção da semântica das classes. Nesta fase já é possível projetar a solução do problema definindo um funcionamento da mesma (passo 4), identificar a semântica dinâmica das classes (passo 5), organizar a estrutura global das classes (passo 6). Neste último passo, é revisto o diagrama estático e, se

necessário, reorganizado a hierarquia dos mesmos, no sentido de melhorar a semântica das classes. No final desta fase um protótipo é apresentado (passo 7).

3. **Identificação dos relacionamentos das classes:** é descoberto como os objetos interagem dentro do sistema, estabelecendo os relacionamentos de instância, de classe, temporais, abstrações entre as classes (passo 8), relacionamentos dinâmicos (passo 9) e relacionamentos pré-pós (passo 10). No passo 8, o diagramas estático é refinado.

4. **Análise comportamental do objeto :** onde são identificadas as restrições de integridade necessárias, fazendo a modelagem do mecanismo de regras ativas do TOM. Esta fase é dividida em cinco passos não descritos aqui, maiores detalhes podem ser vistos em [Furtado93b].

5. **Implementação de classes de objetos:** onde é decidido como cada classe será implementada, obtendo como resultado um refinamento da estrutura da classe e a complementação de cada formulário de classe e de método.

6. **Transformação do esquema conceitual para o esquema lógico:** onde será obtido o esquema lógico do sistema, de acordo com a sintaxe do modelo TOM, a partir de informações contidas nos formulários.

A figura 3.2 ilustra toda a sequencialidade do processo FADO, um processo iterativo e incremental sobre o qual se fundamentam todas as fases da metodologia. As figuras em destaque representam os passos descritos, úteis na definição de um diagrama estático.

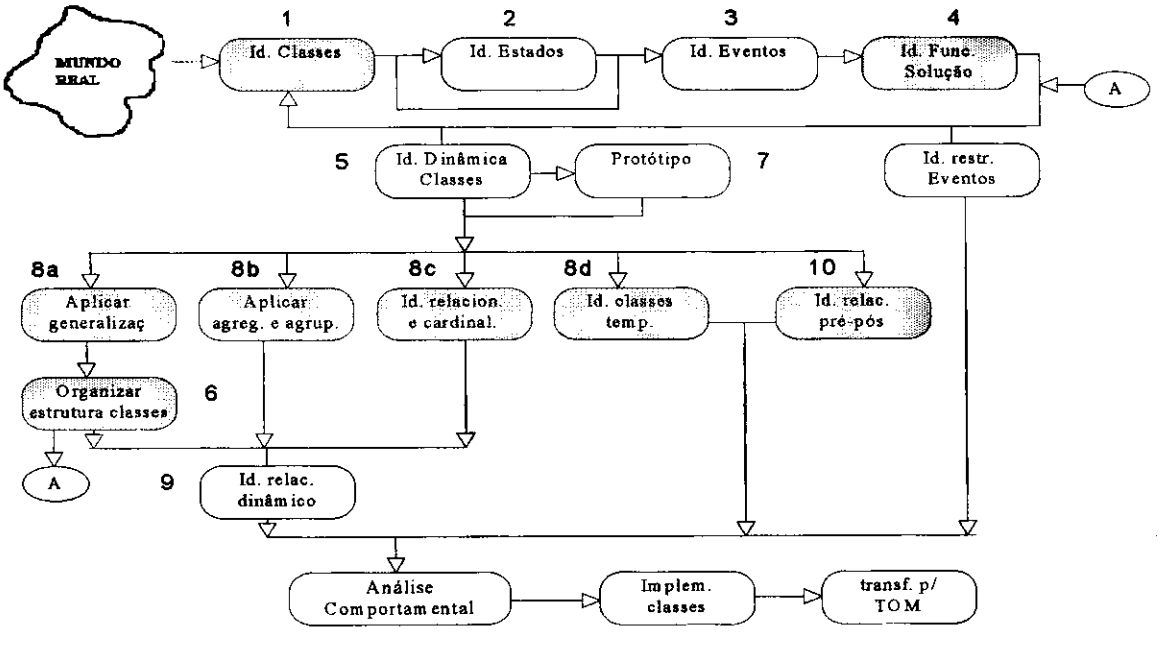


Figura 3.2. Sequencialidade do Processo FADO

3.3.1.2. Metodologia *POKER*

A metodologia ***POKER*** é uma metodologia de desenvolvimento de projeto para sistemas de informação (SI), utilizando um modelo de dados de alto nível que descreve a estrutura (aspecto estático) do SI e uma ferramenta altamente formalizada, a rede de Petri, que descreve o comportamento (aspecto dinâmico) do SI. A metodologia se baseia no método ***OKAY*** [Schiel90], sendo que este último utiliza o modelo ***VODAK*** [Schiel89] e *POKER* utiliza o modelo **TOM**. A metodologia possui uma documentação na forma de diagramas e de tabelas. A metodologia *POKER* é basicamente composta de 8 passos, dividido em duas fases:

1. **Modelo de Requisitos**: onde é feita uma análise dos requisitos de uma aplicação obtendo como resultado um modelo inicial do SI, denominado modelo de requisitos. Ele é obtido através de dois grafos: o grafo **comportamental**(GC), desenvolvido nesta fase (1° passo), é uma rede de fluxo de informações hierárquica sobre a qual a dinâmica da aplicação é descrita, e o grafo **estrutural**(GE), derivado do GC (2° passo), considerado como a arquitetura básica da parte estrutural de um esquema conceitual, sendo que a estrutura de cada objeto componente do esquema e seu relacionamento são extraídos da descrição comportamental da aplicação, representada pelo GC (uma unidade de informação do GC pode corresponder a um objeto ou um relacionamento do GE). Os outros passos desta fase constituem a análise temporal e a análise de restrições.

2. **Esquema Conceitual TOM:** onde é utilizado o modelo de requisitos para o projeto de uma descrição formal e completa do esquema conceitual, projeto da parte computadorizada do SI. Nesta fase, começa o formalismo do modelo de requisitos começando pelo GC sendo transformado em uma rede de Petri (5° passo). Depois, as entidades contidas no GE são classificadas, se tornando um **grafo de estrutura TOM** (6° passo). Esta conversão segue algumas heurísticas sugeridas pela metodologia. Um ECG corresponde ao grafo de estrutura TOM (em destaque na figura 3.3). Outros passos constituem a abstração de métodos, onde toda a informação comportamental de um objeto, que estão dentro das transições da rede de Petri, são encapsuladas dentro do próprio objeto, e a conversão do esquema para a linguagem conceitual do TOM.

A figura 3.3. mostra uma visão geral das duas fases da metodologia *POKER*:

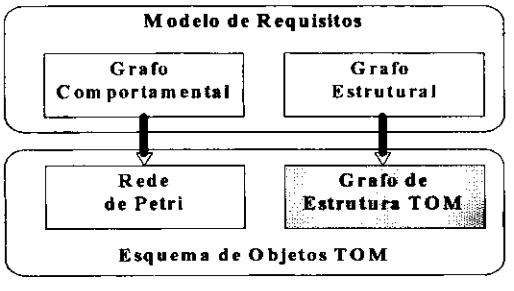


Figura 3.3. As duas fases da metodologia *POKER*

3.3.2. Representação Gráfica de um ECG

Corresponde aos componentes gráficos que constituem um ECG provenientes do diagrama estático, da metodologia FADO ou do grafo de estrutura TOM, da metodologia *POKER*. Será introduzido um exemplo de um ECG correspondente a um sistema de agência de empregos com o seguinte funcionamento:

A agência de empregos presta serviços aos seus clientes, que são as empresas, procurando empregados para elas. Toda pessoa, antes de ser empregada, é cadastrada como candidata a um emprego. A agência obtém informações cadastrais dos candidatos, além de informações adicionais, tais como salário de preferência, etc. Assim que as empresas fornecerem certas informações (tais como: projetos cujas equipes necessitam de novos empregados, número de vagas disponíveis nas empresas, salário médio da equipe a ser formada, entre outras) às agências, estas analisam os candidatos cadastrados para que possam encaminhar empregados qualificados de acordo com a demanda da empresa. Os candidatos que se adequarem às propostas de empregos fornecidas, serão selecionados para se tornarem novos empregados e já são encaminhados para trabalharem em uma empresa específica e dentro de uma equipe de projeto específica. A figura 3.4 ilustra a representação gráfica deste sistema de agência de empregos, por simplicidade, somente alguns relacionamentos foram representados visualmente. A tabela 3.1 mostra as informações sobre todo os relacionamentos contidos nesta representação.

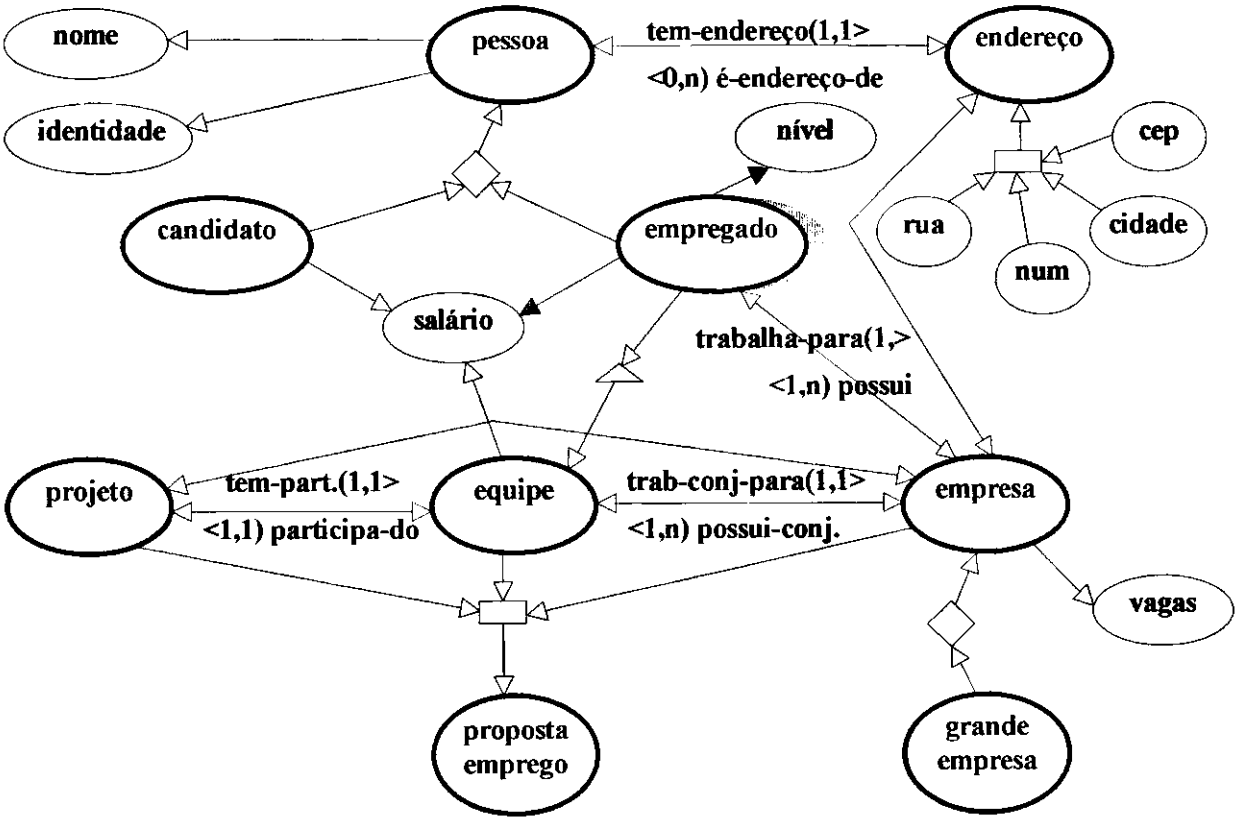


Figura 3.4. Exemplo de um ECG Completo

Classes	Tipo Relac.	Nome/Card Rel. (=>)	Nome/Card Rel. Inv.(<=)
pessoa/nome	instância	*tem-nome (1,1)*	*é-nome-de (0,n)*
pessoa/identidade	instância	*tem-ident (1,1)*	*é-ident-de (1,1)*
pessoa/endereço	instância	*tem-endereço (1,1)*	*é-endereço-de (0,n)*
empregado/salário	instância	*tem-salário (1,1)*	*é-salário-de (0,n)*
empregado/nível	instância	*tem-nível (1,1)*	*é-nível-de (0,n)*
empregado/empresa	instância	*trabalha-para (1,1)*	*possui (1,n)*
empresa/vagas	instância	*tem-num-vagas (1,1)*	*é-num-vagas-de (0,n)*
equipe/salário	instância	*tem-sal-médio (1,1)*	*é-sal-médio-de (0,n)*
equipe/empresa	instância	*trabalha-conj-para (1,1)*	*possui-conj (1,n)*
equipe/projeto	instância	*participa-do (1,1)*	*tem-participantes(1,1)*
empresa/projeto	instância	*utiliza(0,n)*	*é-utilizado-pela (1,n)*
candidato/salário	instância	*salário-pretendido(1,1)*	*pretendido-pelo(0,n)*

Tabela 3.1. Relacionamentos do ECG da figura 3.4

As classes são representadas como **nodos**. As classes não primitivas são representadas com nodos de notação mais destacada que as classes primitivas. As classes temporais são representadas como nodos "sombreados" (ex: *empregado)*. Existe uma outra notação não ilustrada no ECG do exemplo, denominada nodo "duplicado". Ela existe somente em ECG's modulares e será definida na próxima seção. Denominaremos, durante o decorrer de todo este trabalho, as classes primitivas de **atributos** e as não primitivas somente de **classes**.

Os relacionamentos são representados como **ligações**. Os relacionamentos temporais são representados como ligações hachuriadas (ex: *tem-salário)*. Todos os relacionamentos possuem o seu correspondente inverso, porém, por questões de simplicidade, as ligações com classes primitivas não apresentam o seu correspondente inverso. O tipo e a cardinalidade do relacionamento (e o seu relacionamento inverso), são apresentados na tabela 3.1.

O ECG utiliza representações visuais distintas para as diferentes hierarquias de abstração. A generalização/especialização é representada como **losango**. A agregação é representada como **quadrado**, e o agrupamento, representado pelo **triângulo**. Considerações sobre herança e categorias de classificação não são explicitamente representadas no ECG.

Existem algumas restrições implícitas que aparecem no ECG, através das cardinalidades dos relacionamentos. Por exemplo, a restrição de que um empregado trabalha em exatamente uma empresa é representado através das cardinalidades mínima e máxima (1,1) do relacionamento *trabalha-para* entre a classes *empregado e empresa*, ou a informação de que não existe restrição ao número de projetos por

representação, que se refere à modelagem da aplicação dentro do modelo THM, sendo que este modelo foi substituído pelo modelo TOM.

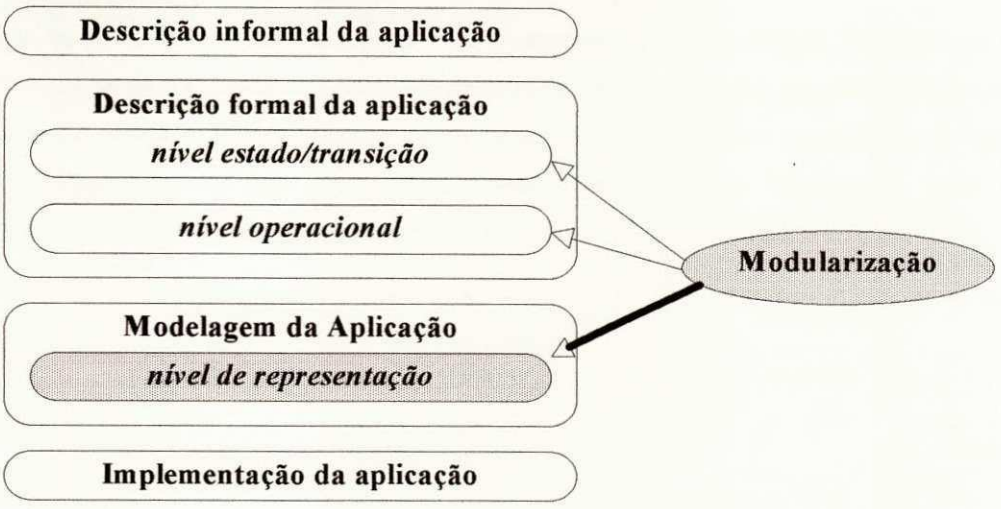


Figura 3.5. Processo de Especificação com níveis e módulos

A construção de um ECG modular é dividido em três passos:

1. **Construção**: indica como um módulo é derivado a partir de outros. A construção dos módulos se efetua através dos seguintes contrutores específicos: o construtor de **combinação**, que consiste na formação de módulos **iniciais**, onde a granularidade de tais módulos depende do contexto; e os contrutores restantes, que correspondem às abstrações do modelo TOM: **generalização, agrupamento e agregação**, derivando novos módulos que contém as classes geradas por essas abstrações. Vale ressaltar novamente que as propriedades dos módulos derivados são herdadas dos módulos originais.

2. **Expansão**: consiste na definição de novas propriedades nos módulos derivados em adição às propriedades que foram herdadas dos módulos originais.

3. ***Hiding*: consiste na eliminação, dentro dos módulos derivados, de certas propriedades que foram herdadas dos módulos originais e que não serão necessárias dentro do contexto dos módulos derivados. Tais propriedades não serão visíveis e nem acessíveis nos módulos derivados.

Iremos agora definir uma nova notação gráfica, o nodo "duplicado". Este nodo representa a classe que está sendo compartilhada entre módulos diferentes. É como se a classe que foi definida em um módulo fosse "distribuída" para outros módulos, derivados ou não. Em cada módulo que a compartilha, ela apresenta somente propriedades de interesse do mesmo. Esta característica é efetuada através do processo de "eliminar" as propriedades que não são necessárias no contexto do módulo.

sistema analisa uma proposta de emprego sempre sobre uma determinada empresa, projeto e equipe específicos. As únicas propriedades herdadas dos outros módulos foram os relacionamentos entre as classes componentes.

Os módulos *Equipe, Equipe-grande-empresa e Proposta-emprego* são ilustrados na figura 3.7:

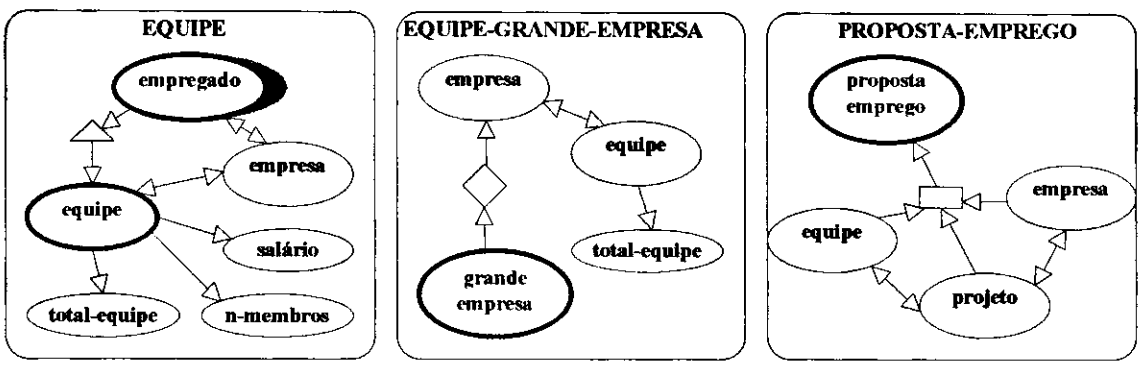


Figura 3.7. Módulos Gerados pelas Abstrações do Modelo TOM

3.3.4. Acesso a um ECG

Após a criação de um ECG, ele será adicionado a uma biblioteca de esquemas dentro do ambiente ConTOM, para futuramente poder ser utilizado. O acesso a um ECG é feito em dois passos: 1) via estrutura modular e 2) via diagrama. Se o ECG não é modular, o primeiro passo é suprimido.

3.3.4.2. Acesso via Estrutura Modular

Inicialmente é apresentado uma organização modular do ECG através da interdependência dos módulos. O usuário poderá ter acesso a qualquer um dos módulos desta estrutura simplesmente selecionando-o. A partir daí, o acesso se torna via diagrama.

3.3.4.1. Acesso via Diagrama

Esta via de acesso se aplica tanto a um ECG não modular, onde é apresentado um ECG similar ao ECG da figura 3.4 com uma tabela similar à tabela 3.1, como também a um ECG modular, onde é apresentado um sub-ECG de um módulo específico. Informações necessárias ao usuário que não estão explícitas no ECG podem ser obtidas através da seleção de cada componente gráfico do mesmo.

A figura 3.8 mostra a organização dos módulos da agência de empregos e um módulo mais detalhado (módulo *Proposta-emprego*):

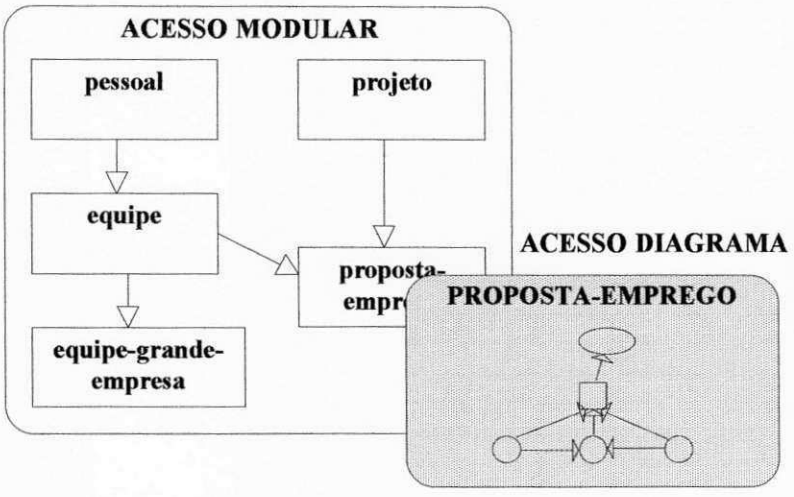


Figura 3.8. Os Dois Níveis de Acesso

Capítulo 4

Especificação de Consultas em ConTOM

4.1. Introdução

A maioria das interfaces gráficas de BD's, que existem atualmente, especificam consultas através da manipulação direta dos componentes que fazem parte do esquema conceitual e que são representados graficamente. Tais componentes gráficos representam classes ou relações, dependendo do modelo sob o qual a interface se fundamenta. Uma consulta a uma base de dados identifica uma ou mais classes (ou relações) que são o escopo de acesso da consulta e um conjunto de condições específicas. O resultado desta consulta é o conjunto de instâncias das classes identificadas (ou tuplas das relações identificadas) e que estão relacionadas ao contexto da consulta, de acordo com aquelas condições.

No ambiente ConTOM, a especificação de uma consulta será executada sobre um ECG (modular ou não), através de mecanismos visuais denominados **Operadores Gráficos de Consulta - OGC's.** Cada OGC manipula classes de objetos e seus relacionamentos, componentes de um ECG, produzindo novas classes de objetos derivadas das classes originais. O processo de recuperação de uma consulta consiste na geração de um novo sub-esquema, denominado **Esquema de Resposta - ER,** com a mesma estrutura do ECG original, podendo ser utilizada como operando em outras consultas. Todo esse processo de especificação possui um alto poder semântico, através do modelo TOM, facilitando a interação visual com o usuário.

Dentro do âmbito de consultas, o ambiente ConTOM possui as seguintes características: *feedback* gráfico a cada ação sobre o OGC, suporte de uma consulta executada sobre um ECG modular, operações de conjunto, consultas temporais, além de possuir uma sintaxe formal para os OGC's. Apesar de apresentar as características acima, ConTOM não suporta quantificadores e consultas recursivas.

Este capítulo irá definir a sintaxe e semântica de especificação de uma consulta pelos OGC's. Não é de nosso interesse discutirmos aqui formas eficientes de implementar estas consultas. O capítulo é dividido em duas partes: a primeira parte define informalmente os OGC's disponíveis em ConTOM. A definição de um OGC será feita nos níveis intensional e extensional, juntamente com exemplos práticos que os utilizam para uma melhor compreensão dos conceitos. Esta primeira parte será a base para a semântica formal dos OGC's, descrita na segunda parte.

4.2. Descrição Informal dos OGC's

Os Operadores Gráficos de Consulta (OGC's) são mecanismos visuais de consulta que se baseiam na técnica da **manipulação direta** [Schneiderman83] sobre os ECG's. Antes de descrevermos os OGC's com mais detalhes, vamos considerar o processo de especificação de uma consulta dividido em três fases, segundo [Catarci93]:

1. O usuário seleciona a parte do BD em que ele quer operar.
2. A partir da parte selecionada, o usuário, define as relações ou condições, de forma a produzir o resultado da consulta.
3. O resultado da consulta é visualizado.

O processo de especificação de consulta no ambiente ConTOM inclui as duas primeiras fases. Na primeira fase, aplica-se um único OGC, denominado ***seleção,*** e na segunda fase, aplicam-se os OGC's restantes, não importando a ordem de aplicação dos mesmos. Todo o processo é efetuado da seguinte forma: a cada OGC realizado, obtém-se um novo módulo, o próximo OGC será aplicado sobre este último módulo gerado. Este processo continua até serem executados todos os OGC's necessários para a realização de uma consulta específica. Note que o processo de especificação é similar ao processo de modularização de um ECG. Um OGC executado, assim como um construtor no processo de modularização, gera um novo módulo. A diferença entre os dois é que um construtor se aplica a qualquer um dos módulos enquanto que um OGC é sempre aplicado ao último módulo gerado. Outro aspecto similar é que alguns OGC's correspondem aos operadores de construção correspondentes às abstrações do modelo TOM. Também há uma analogia com a álgebra relacional em que os operadores se aplicam a relações para gerar novas relações.

Os OGC's estão divididos em dois tipos: OGC's básicos e OGC's temporais. Além da definição informal dos OGC's, a utilização dos mesmos será ilustrada através de exemplos de consultas referentes ao ECG do sistema de agência de empregos.

4.2.1. OGC's Básicos

Correspondem ao OGC da primeira fase, denominado de *seleção* e aos OGC's correspondentes às abstrações do modelo TOM. Os OGC's básicos podem ser aplicados também à classes ou relacionamentos temporais, neste caso, ConTOM considera somente o estado atual do BD.

4.2.1.1. OGC *Seleção*

Consiste na escolha de um subesquema conceitual através da "navegação" entre as classes componentes de um ECG e que estão envolvidas dentro do contexto da consulta. Em [Bertino92] é declarado que a capacidade de "navegar" entre as estruturas dos objetos é uma importante característica das linguagens de consulta orientadas a objeto, desde que modelos de dados orientados a objetos permitem a definição de objetos como agregados de outros objetos. Esta capacidade corresponde à noção de junções em linguagens relacionais, com a diferença que na primeira, a junção entre classes é implícita, e na última os atributos de junção devem ser especificados.

O OGC *seleção* se processa da seguinte forma: o usuário vai selecionando as classes de interesse para a consulta até estabelecer um "caminho" correspondente a uma consulta elementar. Caso o usuário queira formar "caminhos" distintos, deve ser aplicado novamente o OGC *seleção* sobre as classes envolvidas neste novo caminho. Para uma melhor compreensão, a figura 4.1 ilustra duas alternativas de seleção sobre classes e o resultado de cada uma (este resultado aparece no módulo de consultas):

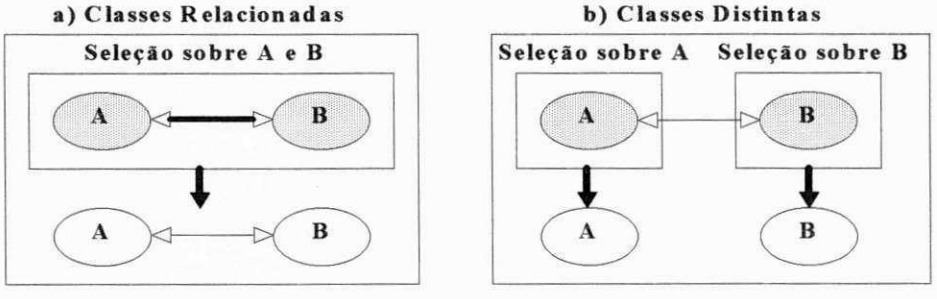


Figura 4.1. Duas Alternativas de Seleção

Na figura 4.1, as duas alternativas representam dois "caminhos" distintos. Em (a), o usuário deseja que as classes A e B estejam conectadas, desta forma, ele aplica uma *seleção* sobre as duas classes. Em (b), o usuário deseja que as duas classes sejam distintas, é necessário, portanto, aplicar a *seleção* duas vezes, uma para cada classe. Na escolha da primeira alternativa, as classes devem estar conectadas por relacionamentos comuns ou hierárquicos (os relacionamentos são automaticamente selecionados).

A nível extensional, quando o operador *seleção* é aplicado a duas classes A e B relacionadas, serão recuperadas as instâncias de A e B que estão **associadas** entre si pelo relacionamento (as instâncias não

relacionadas não serão recuperadas). Na escolha da segunda alternativa, quando o operador *seleção* é aplicado a uma classe A (ou B) "isolada", serão recuperadas **todas** as suas instâncias.

Caso o usuário queira selecionar classes que não estejam diretamente relacionadas, as classes intermediárias que as ligam são automaticamente selecionadas. Na existência de dois ou mais relacionamentos entre as classes envolvidas no "caminho", o usuário deve também selecionar o relacionamento desejado, como ilustra a figura 4.2 (a). Na figura 4.2 (b), a classe B está mais destacada só para indicar que ela foi selecionada automaticamente, pois ela representa uma classe intermediária entre as classes A e C. Havendo mais de um caminho envolvendo classes intermediárias distintas, a escolha do caminho será determinada pela seleção das classes intermediárias do caminho correto, como mostra a figura 4.2 (c):

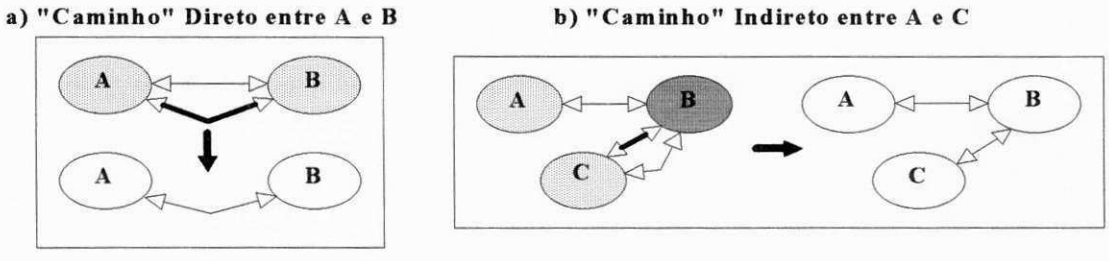


Figura 4.2 (a, b). Duas Alternativas de Seleção sobre Relacionamento

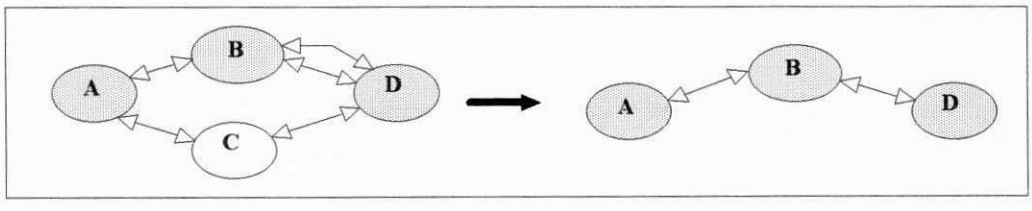


Figura 4.2 (c). Seleção sobre Classe Intermediária

O processo de seleção também possibilita o compartilhamento de classes envolvidas em um ou vários "caminhos" juntamente com classes isoladas, como ilustra a figura 4.3. Na figura, a classe B foi compartilhada por dois caminhos. Desta forma, a classe é indexada no módulo resultante, indicando que as suas instâncias selecionadas são dependentes do contexto de cada "caminho".

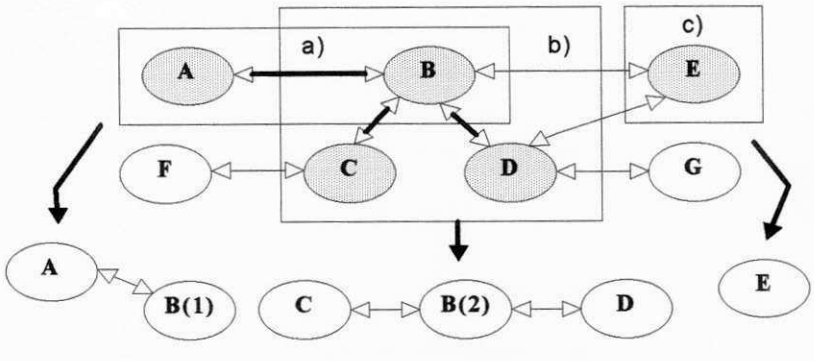


Figura 4.3. Três Caminhos Distintos

O procedimento de indexação de uma classe é utilizado também em consultas que utilizam relacionamento reflexivo, ou melhor, relacionamento que associa uma classe consigo mesma. A consulta "*Recupere todos os empregados e o seus gerentes*" é um exemplo de relacionamento reflexivo, considerando que gerente também é um empregado. A figura 4.4 ilustra o procedimento do usuário e o resultado, dentro do exemplo acima.

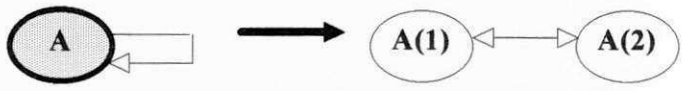


Figura 4.4. Classe com Relacionamento Reflexivo

O OGC *seleção* é o único OGC que pode ser aplicado sobre um ECG completo ou modular. Em um ECG completo, o usuário "navega" sobre as classes componentes, enquanto que em um ECG modular, o usuário somente "navega" sobre as classes componentes, se percorrer os vários módulos que contém as mesmas (ver cap. 3, seção 3.3.3). Independente da estrutura de um ECG, o resultado será o **mesmo**: um *módulo de consultas* contendo o esquema com as classes selecionadas. Este módulo define um contexto de consulta e será a base para a segunda fase de especificação sobre a mesma. Para uma melhor compreensão do processo de seleção, vamos examinar o seguinte exemplo, referente ao sistema de agência de empregos:

Exemplo 1. "Recupere todos os nomes dos empregados acima de 21 anos que trabalham em empresas de Campina Grande no projeto XXX".

Vamos considerar primeiro o ECG não modular da figura 3.4. Sobre este ECG, o usuário seleciona as classes de interesse envolvidas no contexto da consulta. O estado do ECG é correspondente à figura 4.5 (as classes e os relacionamentos selecionados estão em destaque na figura). O resultado deste processo será o *módulo de consultas* da figura 4.6.

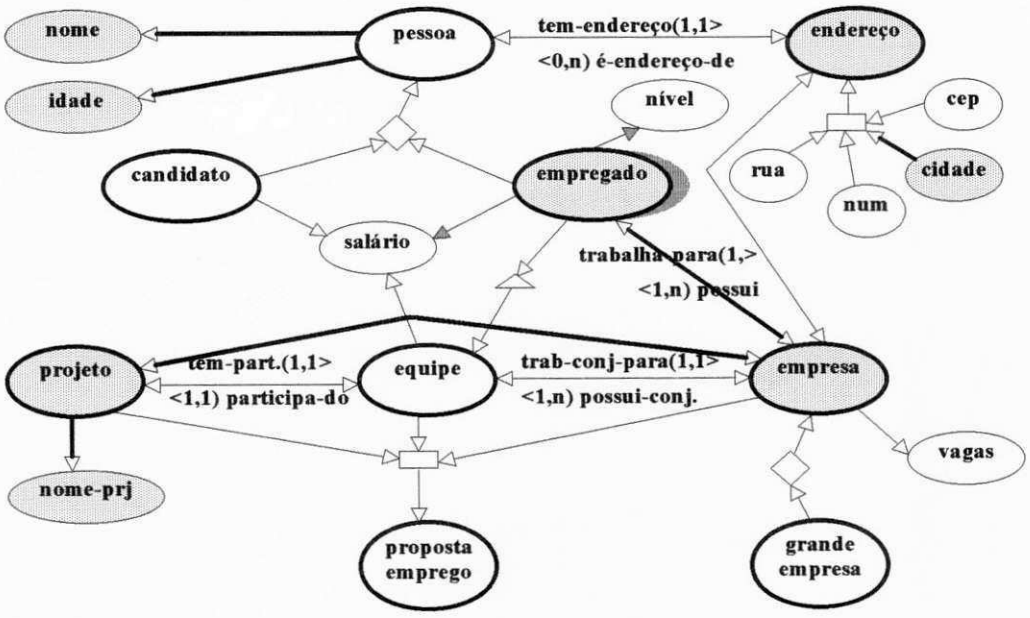


Figura 4.5. Processo de Seleção em um ECG Completo

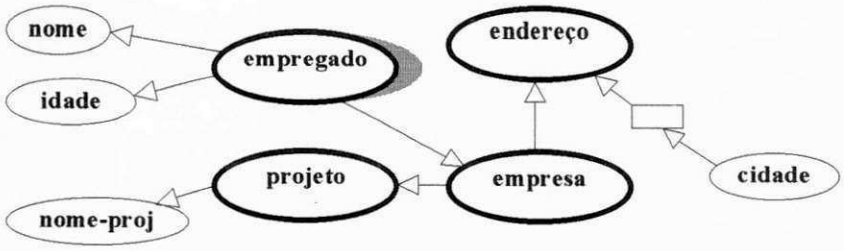


Figura 4.6. Módulo de Consultas Resultante

Existem alguns aspectos a serem analisados no *exemplo 1*. Um aspecto é que a ordem da seleção das classes não é importante. Outros aspectos serão analisados mediante a definição de algumas regras necessárias ao processo de seleção. O processo de seleção só funciona perante o cumprimento destas regras, descritas a seguir.

***Regra 1**: Seleção dupla de uma classe não primitiva implica na seleção de todos os atributos associados a ela, diretamente ou por herança (associados às suas superclasses).*

Se houvesse a seleção dupla da classe *empregado*, todos os seus atributos seriam também selecionados.

***Regra 2:** Seleção de classes que devem estar conectadas implica na seleção automática do relacionamento entre elas.*

O usuário deseja que as classes *empregado* e *empresa* estejam conectadas, desta forma, houve a seleção automática do relacionamento *trabalha-para*.

***Regra 3**: Seleção de classes que devem estar conectadas, porém não estão relacionadas diretamente, implica na seleção automática de classes intermediárias que as ligam.*

A relação entre as classes *cidade* e *empresa* só pode ser expressa via uma classe intermediária, neste caso, a classe *endereço* foi automaticamente selecionada.

***Regra 4**: Seleção de classes que devem estar conectadas e que possuem mais de um relacionamento entre elas, implica na seleção do relacionamento desejado, feito pelo usuário.*

Se existisse um relacionamento *é-presidente-de* entre as classes *empregado* e *empresa*, o usuário deve escolher entre este relacionamento e o relacionamento *trabalha-para*.

Vamos considerar um aspecto que se refere à seleção de classes conectadas hierarquicamente, através das abstrações do modelo TOM, permitindo consultas tipo: "*encontre os empregados da equipe cujo salário*

médio > xxx" ou *"encontre o nome das pessoas com salário de preferência > xxx, se elas são candidatas"* ou *"encontre as propostas de emprego de grandes empresas"*, etc. Neste tipo de relacionamento, os predicados correspondentes às abstrações são mantidos na interpretação do módulo de consultas, isto é, se no módulo de consultas, por exemplo, existem duas classes conectadas por agrupamento, o predicado *é-elemento* entre as classes é mantido, onde cada instância de uma classe é elemento de um conjunto de instâncias da classe agrupada. [Bertino92] define este predicado como um predicado *membership* ou de inclusão.

Agora vamos considerar o processo de seleção do *exemplo 1* sobre o ECG modular da figura 3.6 acrescentado de um novo módulo, o módulo *Endereço*. O usuário utiliza o mesmo procedimento do exemplo, só que percorrendo os módulos *Pessoal, Projeto e Endereço*, como mostra a figura 4.7. O resultado será o mesmo módulo de consultas da figura 4.6.

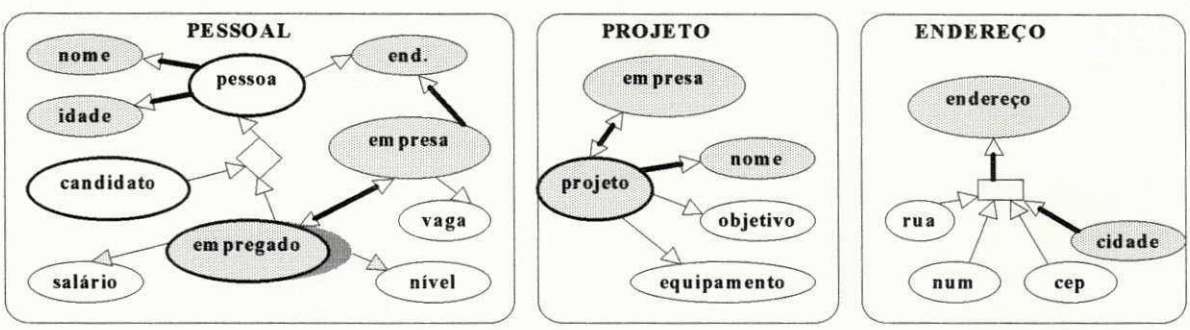


Figura 4.7. Processo de Seleção em um ECG modular

Para finalizar, o processo de seleção que foi descrito é baseado na abordagem utilizada na linguagem OQL [Alashqur89] e no modelo de consultas de BDOO's [Kim89]. A linguagem OQL possui um operador de *associação* similar ao OGC *seleção*. Uma consulta OQL retorna um sub-BD de um BD específico. Um sub-BD consiste de duas partes: uma associação intensional e um conjunto de associações extensionais. Uma associação intensional é representada como um conjunto de classes e suas associações e atributos descritivos. Uma associação extensional é uma rede de instâncias e suas associações que pertencem às classes da associação intensional correspondente. O conjunto de associações extensionais pode ser representado na forma de um diagrama extensional. Um exemplo de uma associação intensional é o ECG da figura 4.5. Um possível diagrama extensional correspondente a este ECG, considerando somente as classes *empregado*, *empresa* e *projeto*, é ilustrado na figura 4.8.

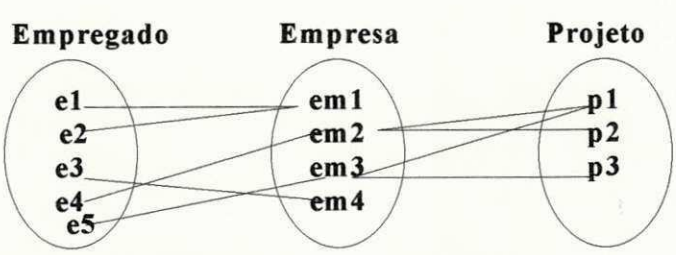


Figura 4.8. Exemplo de um Diagrama Extensional

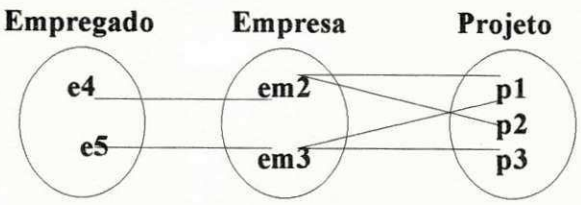


Fig 4.9. Diagr. Extensional do Módulo de Consultas

Após a aplicação do OGC *seleção* sobre o ECG da figura 4.5, o diagrama extensional do módulo de consultas gerado (figura 4.6) é ilustrado na figura 4.9 (a figura não ilustrou a classe *endereço* e nem os atributos). Note que as instâncias *em1* e *em4* não fazem parte do diagrama extensional, pois a seleção só vai recuperar as instâncias de *empresa* associadas à instâncias de *projeto*. Como consequência, as instâncias *e1*, *e2* e *e3* não serão recuperadas.

O modelo de [Kim89] é mais limitado pois a junção é restrita à ligação atributo-domínio entre classes especificadas em esquemas orientados a objetos, ou seja, a junção é efetuada somente entre pares de classes *ci* e *cj*, onde *cj* é o domínio de um atributo de *ci*. Consultas que exigem uma ligação inversa de atributo-domínio entre este par de classes não podem ser feitas. ConTOM suporta este último tipo de consulta, já que o modelo TOM adota o conceito de relacionamento e relacionamento inverso entre as classes.

4.2.1.2. OGC *Especialização*

Consiste na seleção de uma classe, com o objetivo de criar uma subclasse de instâncias que satisfazem uma restrição baseada em um atributo associado diretamente ou não a esta classe. O critério de especialização pode ser definido como um predicado ou uma combinação booleana de predicados usando conectivos lógicos *and, or* ou *not* sobre os atributos da classe inicial selecionada.

Um predicado em ConTOM é expresso da seguinte forma: *Operando1 Operador Operando2*. *Operando1* é um **atributo** direto ou indireto da classe a ser especializada. Define-se um atributo indireto, aquele que está associado à classe a ser especializada, através de uma ou mais classes intermediárias. *Operador* refere-se aos **operadores de comparação** $=, \neq, >, \geq, <, \leq, \in, \notin, \subset, \not\subset$ e *Operando2* pode ser de três tipos: uma **constante**, um elemento de uma **lista de valores** e um **atributo** direto ou indireto. A idéia da lista de valores foi retirada do trabalho de [Kuntz89], onde ele considera que este tipo de facilidade contribui para uma formulação "cooperativa" das consultas, evitando o resultado "vazio" provocado pela entrada de um valor desconhecido em uma condição (por exemplo, o atributo *cor* possui uma lista de possíveis valores).

A consulta do *exemplo 1* possui três restrições: os empregados acima de 21 anos, as empresas de Campina Grande e o projeto XXX. O usuário, a partir do módulo da figura 4.6, aplica o OGC *especialização* sobre as classes a serem especializadas juntamente com os seus atributos (diretos ou não),

dentro da sequência classe-atributo, como ilustra a figura 4.10a. O resultado será um novo módulo ilustrado na figura 4.10b (note que as novas subclasses herdaram todos os atributos das suas superclasses).

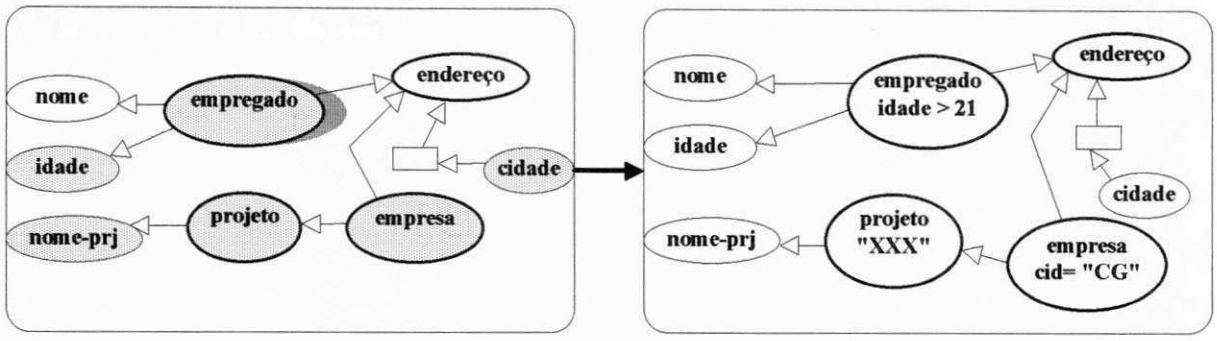


Figura 4.10a. Processo de Especialização

Figura 4.10b. Módulo após Especialização

Note que, no exemplo, a restrição referente às empresas de Campina Grande tem como *Operador1* um atributo indireto, a classe *cidade*. Outro aspecto a ser apresentado na figura acima é que a classe *empregado-idade>21* é uma classe não temporal, indicando que a restrição só se aplica às instâncias atuais da classe. Considerando o diagrama extensional da figura 4.9, vamos supor que todos os empregados tem idade acima de 21 anos, a empresa que se localiza em Campina Grande corresponde a instância *em2*, e o projeto XXX refere-se à instância *p1*. O novo diagrama extensional referente ao módulo da figura 4.10b é ilustrado na figura 4.11.

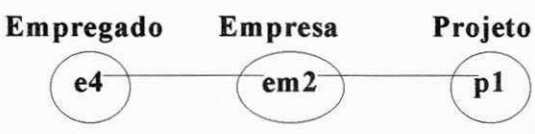


Figura 4.11. Diagrama Extensional

Existe uma outra forma de especificar a mesma consulta, possibilitando ao usuário uma forma de interação mais flexível. Em vez de restringir as três classes distintas, o usuário pode aplicar o OGC *especialização* somente sobre *empregado*, construíndo os termos, *p1: idade* > 21; *p2: cidade(empresa)* = "CG" (o usuário seleciona a sequência *empregado-empresa-cidade*), onde *operando1* é o atributo indireto *cidade*; e *p3: nome-proj(projeto(empresa))* = "XXX" (o usuário seleciona a sequência *empregado-empresa-projeto-nome-projeto*), onde *operando1* é o atributo indireto *nome-proj*.

Esta outra forma de especialização é ilustrado na figura 4.12, apresentando quatro estados do mesmo módulo. Note que, após o usuário especificar um termo, ele aparece ao lado do atributo que foi especializado. Após a construção destes predicados, ConTOM, por *default*, assume o predicado ***p1 and p2 and p3*** como critério de especialização de *empregado* (o usuário pode alterar o predicado *default*). O resultado será um novo módulo ilustrado na figura 4.13.

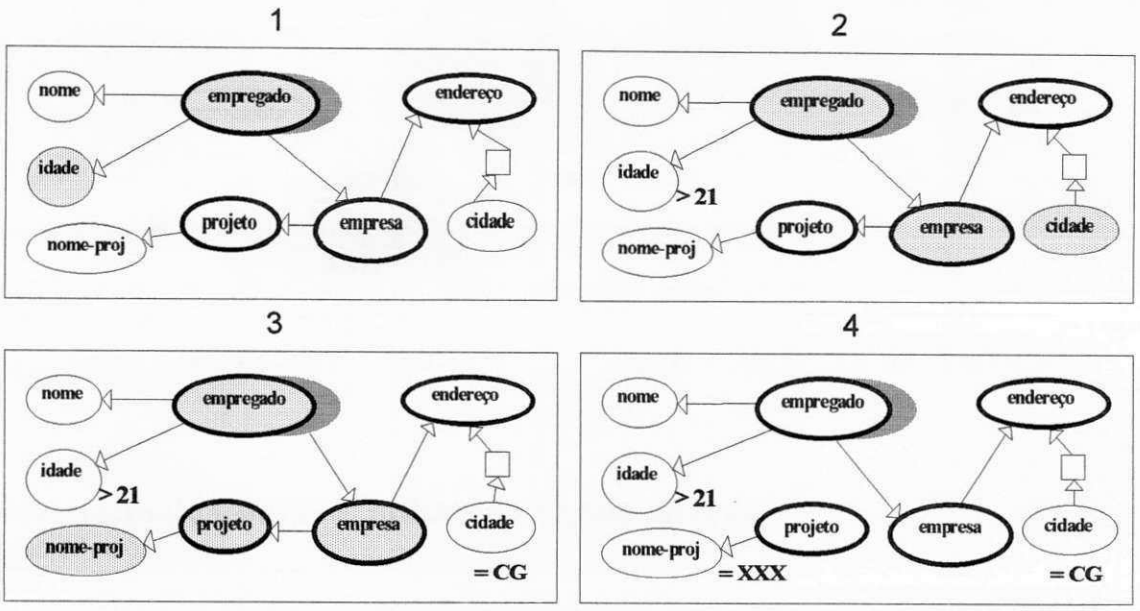


Figura 4.12. Processo de Especialização

Figura 4.13. Módulo resultante da Especialização

Vamos introduzir outro exemplo que envolve classes que compartilham o mesmo atributo:

Exemplo 2. "Recupere os empregados de nível superior que possuem salário maior que o salário médio da equipe a que ele pertence".

O processo de seleção é realizado sobre o módulo da figura 4.5. Houve uma seleção dupla sobre *empregado*, portanto, os seus atributos, mais os herdados da sua superclasse (*pessoa*), vão aparecer no módulo de consultas ilustrado na figura 4.14a. A partir deste módulo, é aplicado o OGC *especialização* sobre *empregado*, formando os termos, *p1: nível = "superior* e *p2: salário > salário(equipe)* (para o *Operando1*, o usuário seleciona a sequência *empregado-salário*, para o *Operando2*, o usuário seleciona a sequência *empregado-equipe-salário*, onde *salário-médio* é o *Operando2* que foi definido como um atributo indireto. O usuário procede da mesma forma apresentada na figura 4.12. Após a construção destes predicados, ConTOM, novamente, por *default*, assume o predicado ***p1 and p2*** como critério de especialização de *empregado*. O novo módulo é ilustrado na figura 4.14b:

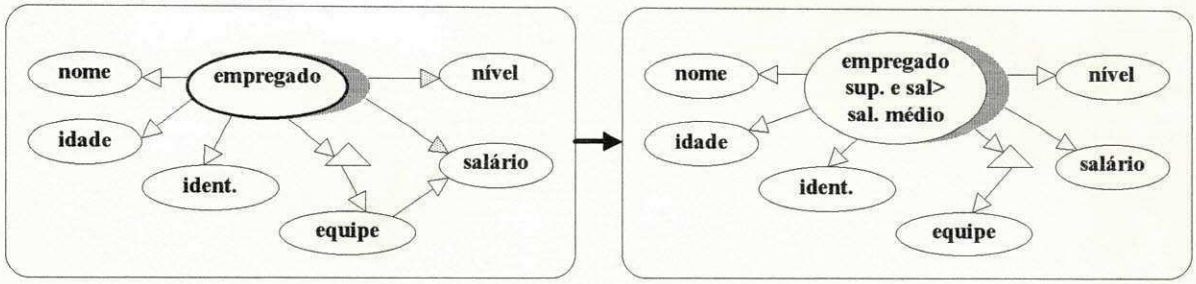


Figura 4.14a. Módulo de Consultas **Figura 4.14b. Módulo após Especialização**

⇒ Operações de Conjunto utilizando *Especialização*:

Uma semântica **precisa** para os operadores de conjunto sobre os objetos complexos ainda não foi desenvolvida [Rundensteiner92]. Em ConTOM, a abordagem sobre operações de conjunto não é a mesma explorada em [Rundensteiner92], que divide o relacionamento subclasse/superclasse em dois tipos: subconjunto/superconjunto, baseado na identidade dos objetos, e subtipo/supertipo, baseado no estado dos objetos. A derivação de uma classe por uma operação de conjunto, em [Rundensteiner92], considera de forma independente os dois relacionamentos. ConTOM utiliza **especialização múltipla**, um caso particular da especialização, para implementar as operações de conjunto **diferença e intersecção** entre classes (o operador **união** será utilizado na generalização, descrito a seguir).

A especialização múltipla consiste na seleção de duas classes com o objetivo de gerar uma subclasse com instâncias que satisfaçam a um operador de conjunto sobre as classes iniciais. A diferença do predicado em relação à especialização comum é que *Operando1* e *Operando2* são as classes a serem especializadas e *Operador* refere-se aos operadores de conjunto "$\cap$" (intersecção) e "-" (diferença). Se aplicado o operador "-", a interpretação da nova classe consiste nas instâncias que pertencem ao *Operando1* e que não pertençam ao *Operando2*. Se aplicado o operador de intersecão, a interpretação da nova classe consiste nas instâncias que pertencem ao *Operando1* e que também pertençam ao *Operando2*. O processo de especialização múltipla, porém, só pode ser aplicada perante o cumprimento das seguintes regras:

***Regra 5**: As classes envolvidas na especialização múltipla devem ser compatíveis.*

Duas ou mais classes são ditas compatíveis se elas fazem parte da mesma hierarquia de generalização, ou melhor, existe um relacionamento *é-um* entre elas ou compartilham este relacionamento com alguma outra classe.

***Regra 6**: Se as classes envolvidas pertencem ao mesmo nível de especialização, o critério não deve ser disjunto.*

Se o predicado entre as classes envolvidas é o operador de diferença, a herança dos atributos só é efetuada sobre a classe que representou o primeiro operando. No caso da intersecção, a subclasse gerada terá (por herança múltipla), todos os relacionamentos das duas superclasses, podendo surgir o conflito.

Agora vamos introduzir um exemplo referente ao nosso ECG da agência de empregos, utilizando especialização múltipla. Iremos substituir a classe *candidato* pela classe *estudante.*

Exemplo 3. "Encontre todas as pessoas de nível superior, que são empregados e estudantes"

Houve uma seleção dupla sobre as classes *empregado* e *estudante*, gerando o módulo de consultas ilustrado na figura 4.15a (note que houve repetição dos atributos das classes selecionadas). A partir deste módulo, o usuário aplica o OGC *especialização* sobre as classes *estudante* e *empregado* (em destaque na figura), formando o predicado *estudante* $\cap$ *empregado* como critério de especialização das duas classes selecionadas, gerando o módulo ilustrado na figura 4.15b. A partir deste módulo, a sequência se processa normalmente, especializando a nova classe gerada pelo nível, etc.

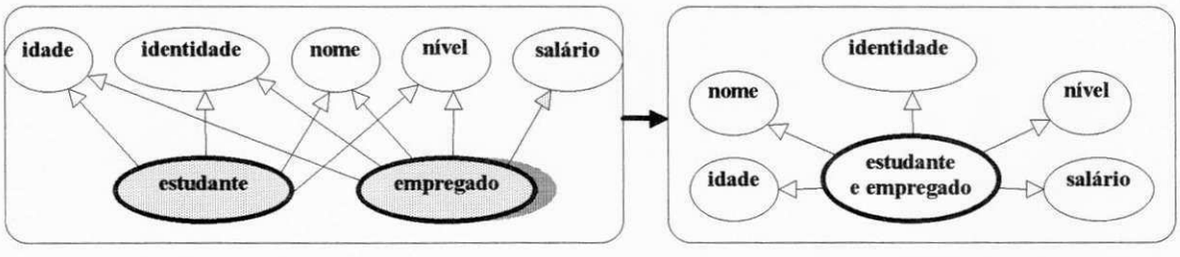


Figura 4.15a. Especialização Múltipla com Intersecção **Figura 4.15b. Módulo após Especialização**

O mesmo procedimento pode ser utilizado para o operador de diferença, em consultas tipo *"encontre todos os estudantes que não são empregados"*, ou vice-versa.

4.2.1.3. OGC *Generalização*

Consiste na seleção de classes com o objetivo de gerar um módulo contendo uma superclasse daquelas classes, associada a atributos que são comuns entre as mesmas. As instâncias da superclasse gerada correspondem à união das instâncias das classes selecionadas. Desta forma, ConTOM utiliza o OGC *generalização,* assim como a especialização múltipla, para implementar uma operação de conjunto, em particular, a **união** ($\cup$).

O processo de generalização é menos restrito que a especialização múltipla no sentido de que as classes envolvidas não necessariamente precisam ser compatíveis, como iremos ver no exemplo referente ao nosso ECG da agência de empregos, utilizando a generalização:

Exemplo 4. "Encontre todas os empregados de nível superior e as empresas de Campina Grande".

Existe uma diferença entre a consulta do *exemplo 4* e uma consulta tipo *"encontre todos os empregados de nível superior que trabalham em empresas de Campina Grande"*. No *exemplo 4* existem dois caminhos diferentes, como consequência, as classes envolvidas nos caminhos não devem estar relacionadas. Desta forma, a *seleção* é aplicada sobre *empregado* e *empresa* separadamente, gerando o módulo de consultas da figura 4.16. A partir deste módulo, pode ser aplicado o OGC *generalização* sobre as classes *empregado* e *empresa* (em destaque na figura), gerando um novo módulo ilustrado na figura 4.17. A partir deste módulo, a sequência se processa normalmente.

Figura 4.16. Módulo de Consultas

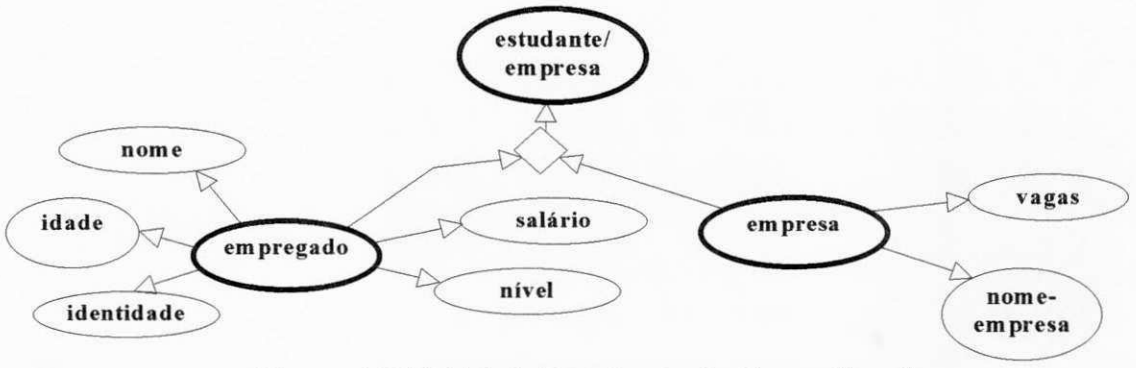


Figura 4.17. Módulo Resultante da Generalização

O processo de especialização pode ser feito a partir do módulo da figura 4.10b, especializando as classes *empregado* e *empresa*, ou poderia ser feito no módulo anterior, e ser aplicado a generalização das classes já especializadas, ressaltando novamente que a ordem dos operadores não é relevante.

4.2.1.4. OGC *Agregação*

Consiste na seleção dos atributos de interesse para o resultado da consulta. O resultado é um novo módulo contendo uma classe agregada cujos componentes são os atributos selecionados.

Para uma melhor compreensão do processo de agregação, vamos considerar o módulo da figura 4.10b do *exemplo 1* adicionado da classe *nome-empresa*. A partir deste módulo, se o usuário deseja recuperar além

do nome dos empregados, o nome das empresas, ele aplica o OGC *agregação* sobre as classes *nome* e *nome-empresa*, ilustrado na figura 4.18a, gerando o módulo da figura 4.18b.

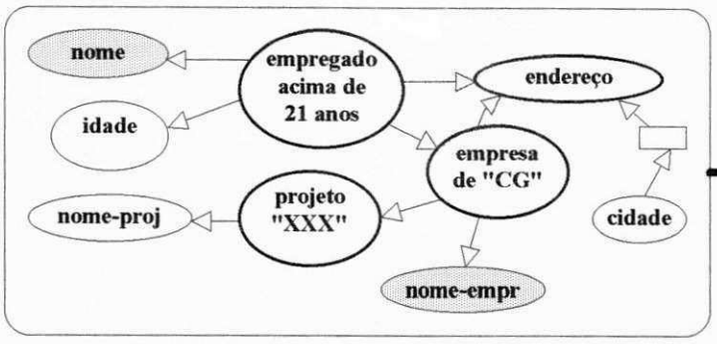


Figura 4.18a. Processo de Agregação

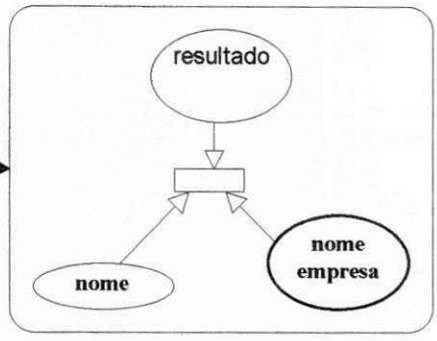


Figura 4.18b. Módulo após Agregação

4.2.1.5. OGC *Agrupamento*

Consiste na seleção de uma classe, cujas instâncias serão agrupadas em função de um determinado parâmetro que corresponde a um valor único ou intervalo de valores de qualquer atributo ou classe relacionada à classe selecionada. O resultado é um novo módulo contendo a classe agrupada juntamente com o parâmetro, ligados por um relacionamento. Este relacionamento é idêntico ao relacionamento entre a classe original e o parâmetro, com exceção da cardinalidade, que se torna adequada à classe agrupada. A classe agrupada equivale ao conceito de **classe parametrizada**, definida em [Abiteboul91], onde ele apresenta uma forma parametrizada de agrupar objetos como uma característica adicional no seu mecanismo de **visões**.

Para uma melhor compreensão do processo de agrupamento, vamos introduzir o seguinte exemplo do nosso ECG, adicionado da classe *departamento* que está associada à classe *empregado*, por um relacionamento comum, e associada à classe *empresa*, por um relacionamento de agrupamento:

Exemplo 5. "Encontre todas os empregados da empresa "xxx". Recupere nome, salário e departamento de cada um. Discrimine empregados por salário e departamento".

A consulta acima possui dois parâmetros de agrupamento (*salário* e *departamento*). O módulo de consultas gerado pelo processo de seleção é ilustrado na figura 4.19a. A partir deste módulo, é aplicado o OGC *agrupamento* sobre a classe *empregado* juntamente com *salário* e *departamento*, na sequência classe-atributo (a ordem dos atributos influirá no resultado), gerando a classe agrupada associada aos parâmetros com o mesmo relacionamento existente entre a classe *empregado-salário* e *empregado-departamento*, alterando somente a cardinalidade de (1,1) para (1,*) do último relacionamento. O módulo resultante está ilustrado na figura 4.19b.

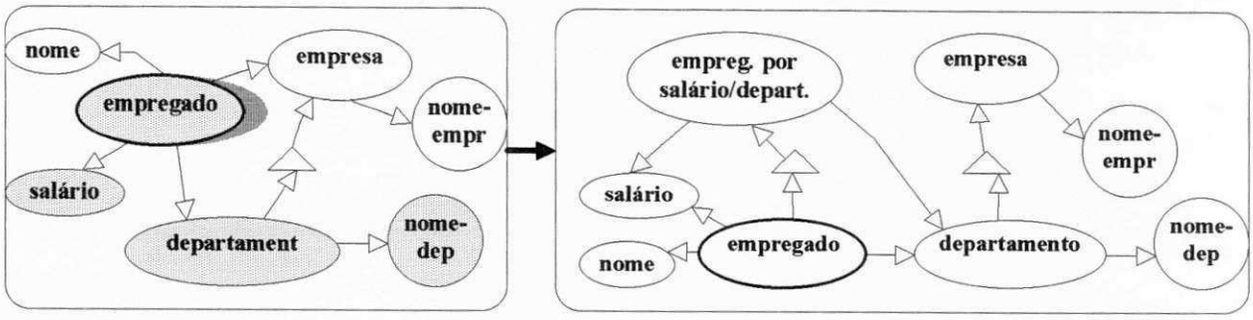


Figura 4.19a. Módulo de Consultas

Figura 4.19b. Módulo Resultante do Agrupamento

4.2.2. Esquema de Resposta

Após a especificação completa de uma consulta, ConTOM gera o **Esquema de Resposta (ER)**, que corresponde ao módulo gerado pelo OGC *agregação* e contém a classe agregada *Resultado*, cujos componentes são os atributos de interesse para o resultado da consulta. O módulo ER mais os outros módulos gerados nas aplicações dos OGC's, compõem um novo ECG, chamado **ECG da Consulta**, que será armazenado na biblioteca de esquemas, podendo ser posteriormente reutilizado. A estrutura deste ECG é modular, cujo módulo inicial corresponde ao módulo de consultas gerado após a seleção sobre o ECG inicial.

A nível extensional, O ER é um conjunto de instâncias, que são agregados de valores dos atributos que fazem parte do resultado da consulta, e que estão associados às instâncias das classes que satisfazem as condições propostas dentro do contexto da consulta.

Considerando o *exemplo1*, modificado para ***"Recupere todos os nomes dos empregados acima de 21 anos que trabalham em empresas de Campina Grande no projeto XXX". Recupere também o nome das empresas"***, o ECG da consulta é mostrado na figura 4.20:

ER da Consulta do Exemplo 1

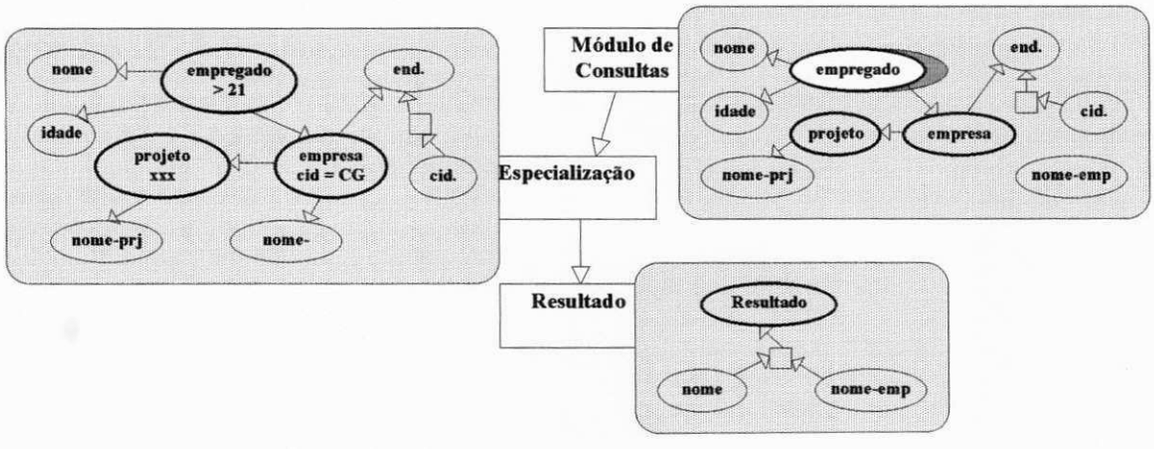


Figura 4.20. Estrutura Modular de um ER

4.2.3. OGC's Temporais

O modelo TOM considera possíveis aspectos temporais dos dados, permitindo a modelagem de aplicações onde o fator **tempo** é determinante. Os OGC's temporais são aplicados sobre os componentes temporais de um ECG: a classe e o relacionamento temporal (relacionamento com valores anteriores). Antes de descrevermos os OGC's temporais com mais detalhes, é necessário uma melhor compreensão das características que são relevantes à modelos de dados temporais, em particular, ao modelo TOM.

4.2.3.1. Conceitos fundamentais para a especificação do Tempo

1. Ordem no Tempo

A definição de uma ordem a ser seguida no tempo é fundamental quando utiliza-se alguma representação temporal [Edelweiss94]. O mais comum é o tempo que flui **linearmente**; isto implica em uma ordenação total entre quaisquer dois pontos no tempo. Definido dois pontos diferentes no tempo t e t', representando a ordem de precedência temporal através do operador "<", uma das seguintes expressões é verdadeira: $t < t'$ ou $t' < t$. Em alguns casos pode ser considerado um tempo **ramificado** no futuro, permitindo a possibilidade de dois pontos diferentes serem sucessores imediatos de um mesmo ponto. Uma última opção de ordenação temporal é considerar o tempo **circular**. Esta forma pode ser utilizada para modelar eventos e processos recorrentes [Maiocchi91]. A ordem no tempo a ser considerada é o tempo linear.

2. Dimensão de Tempo

Existem duas dimensões principais de tempo: tempo **válido**, que é o tempo em que um fato é verdadeiro na realidade modelada e tempo **de transação,** que é o tempo em que um fato é armazenado no BD. O modelo TOM incorpora as duas dimensões.

3. Representação Temporal Explícita e Implícita

A definição de tempo pode ser de forma **explícita**, através da associação de um tempo a uma informação (*timestamping)*, ou de forma **implícita**, através da utilização de uma lógica temporal. Para a representação explícita de tempo, é necessária uma definição de um elemento temporal primitivo, como instante ou intervalo. Um **instante** representa um ponto de tempo particular, e o **intervalo de tempo** é o tempo decorrido entre dois instantes. Nos modelos baseados no intervalo de tempo, o instante é definido como um intervalo muito pequeno e indivisível.

O modelo TOM adota o intervalo como o elemento primitivo na representação explícita, representado pelos instantes *início* e *fim* que compõem este intervalo. Os valores *início* e *fim* são unidades de tempo pré-estabelecidas ou o valor especial *agora* [Schiel93]. Quando um dos instantes é representado por este último valor, em um tempo válido, temos a representação de um intervalo particular cujo tamanho varia com a passagem do tempo [Edelweiss94]. No tempo de transação, o valor de *fim* é utilizado somente se o fato registrado em *início* é falso [Ahn86].

4. Variação Temporal

Considerando o tempo linear, a variação temporal pode ser de três tipos: **discreta**, onde os instantes são isomorfos aos números naturais, ou seja, cada instante possui um único sucessor; **densa**, onde os instantes são isomorfos aos números racionais, ou seja, entre quaisquer dois instantes de tempo, sempre existe outro, e a **contínua,** onde os instantes são isomorfos aos números reais, isto é, ele é denso e diferente dos números racionais, pois não possui espaços e a cada número real corresponde um instante [Jansen94]. A variação discreta é baseada em uma linha de tempo composta de uma sequência de instantes consecutivos que não podem ser decompostos e são de idêntica duração, denominados ***chronons***. O modelo TOM adota o modelo discreto como variação temporal.

5. Granularidade

A granularidade consiste na duração de um *chronon*. As granularidades mais utilizadas são as que fazem parte do sistema de **calendário** (segundo, minuto, hora, dia, mês, ano, etc) e, dependendo da aplicação, podem ser necessárias várias granularidades. Esta capacidade dá ao usuário a facilidade de tratar informações temporais em vários níveis de abstrações [Cavalcanti94]. O modelo TOM utiliza o sistema de calendário gregoriano.

6. Classificação de BD's temporais

Os sistemas de BD's foram classificados, considerando a dimensão do tempo, por *Snodgrass* [Snodgrass85], como: **instantâneos**, que são os BD's clássicos e contém apenas informações sobre os dados válidos atuais; **históricos**, onde à unidade de dado no BD é associado o tempo válido desta unidade de dado, ou seja, o armazenamento das informações varia com o tempo, podendo obter-se o histórico das mesmas; ***rollback,*** onde são armazenados todos os estados passados do BD, utilizando o tempo de transação e os **temporais**, como o modelo TOM, que combinam as propriedades dos sistemas *rollback* e histórico.

4.2.3.2. Representação Temporal dos Dados no Modelo TOM

Em [Brayner94], são caracterizados as propostas de modelos, com representação temporal dos dados, através dos seguintes fatores: modelo de dados suporte, número de dimensões temporais e nível de incorporação do tempo. Considerando este último fator, as dimensões temporais dentro modelo TOM podem ser incorporadas em três níveis: nível ***object timing,*** onde são temporalizados os objetos do mundo real; nível ***property timing***, onde são temporalizados os relacionamentos entre objetos do mundo real e ***schema timing***, onde é registrado o histórico do próprio esquema de BD's [Schiel91]. Os níveis *Object Timing e Property Timing* serão vistos com mais detalhes a seguir.

⇒ Nível *Object Timing*

Este nível é definido por classes temporais, descritas no capítulo 3. Uma classe temporal é declarada como subclasse da metaclasse *TemporalObject,* que possui propriedades e métodos que serão herdados por todos os objetos instâncias da classe temporal declarada. As propriedades são denominadas *history e rollback.*

A propriedade *history* corresponde ao(s) intervalo(s) de tempo durante o(s) qual(is) um objeto foi declarado como instância da classe. Se o fim do intervalo é o valor *agora*, o objeto é instância atual da classe. Se a propriedade *history* contém mais de um intervalo, indica que o objeto foi instância da classe por diversas vezes.

A propriedade *rollback* especifica um intervalo durante o qual o objeto foi conhecido no BD como instância da classe. Este intervalo consiste de dois pontos de tempo: o ponto *entered*, que pode coincidir com o valor de "início" do intervalo de *history*, e o ponto *error*, indicando que o objeto foi erroneamente declarado como instância da classe para o intervalo histórico correspondente. Na maioria das vezes o ponto *error* é um *nil.*

Os métodos vão garantir que os objetos deletados não serão removidos do BD e que ao intervalo de *history* será designado o valor de *fim* corretamente. Abaixo é ilustrado como a metaclasse *TemporalObject* é definida:

metaclass TemporalObject
    instance-of TOM-Class
    instance-relationships
        historyBegin: Timepoint
        historyEnd: Timepoint

```
        rollbackBegin: Timepoint
        rollbackEnd: Timepoint
    events ON delete self at t
        DO let t1 be now
            establish self historyBegin t0
                self historyEnd t1
                undo(delete)
```

⇒ Nível *Property Timing*

Este nível retém os valores anteriores de um relacionamento, descritos no capítulo 3. O relacionamento entre duas classes com relação aos valores das mesmas varia com o tempo. Quando da atualização de um relacionamento entre um objeto e um valor ou outro objeto, o relacionamento com o valor anterior não é removido do BD, seu intervalo de tempo é fechado e o intervalo do novo relacionamento é inicializado (o intervalo considerado aqui é o histórico, referente ao tempo válido).

O *property timing* de um relacionamento de uma classe *c1* a uma classe *c2* é obtido através da substituição da classe *c2* por uma classe agregada, cujas classes componentes são: a própria classe *c2* e uma classe de intervalos de tempo *Ih*, equivalentes ao intervalos históricos, como é mostrado abaixo:

```
class HD
    aggregation of D, TimeInterval explicit
events ON remove self rel <d, <t1, now> >
    DO let t2 be now
        establish self rel <d, <t1,t2> >
    ON establish self rel d
    DO let t1 be now
        establish self rel <d, <t1, now> >
        undo(remove)
```

4.2.3.3. Consultas Temporais

Existe um número considerável de modelos de dados que foram estendidos para poderem tratar com a dimensão temporal, destacando a incorporação do contexto temporal no modelo relacional: [Ariav86], [Clifford85], [Gadia88], [Käfer90], [Lorentzos88], [Navathe87], [Tanzel85]. Além do modelo relacional, existem outras propostas para outros modelos, destacamos os modelos ERAE [Dubois86], TEER

[Elmasri90], ERT [Loucopoulos91], TEMPORA [Theodoulidis91] e OODAPLEX [Wuu93]. Para os modelos orientado a objeto, destacamos os modelos OSAM* [Stanley91], o TDM [Segev87], e o próprio TOM [Schiel91].

As linguagens de consulta também foram estendidas para manipularem o tempo. Diversas possibilidades foram estudadas, entre as quais, a introdução de novos operadores temporais, a inclusão de lógica temporal, dedução sobre o tempo, entre outras. A abordagem mais utilizada foi a extensão da álgebra relacional [Gadia88], [Lorentzos88], [Tanzel85]. Entre as linguagens de consultas que foram estendidas, destacamos o QUEL, extendida para TQUEL [Snodgrass87] e HTQUEL [Gadia88]; o SQL, extendida para TOSQL [Ariav86], TSQL [Navathe87] e HSQL [Sarda90] e o OQL [Alashqur89], estendida para OQL/T [Stanley91].

Em ConTOM, vamos considerar basicamente dois tipos de consultas temporais: consultas ***as-of*** e consultas ***walk-through*** [Ahn86]. As consultas *as-of* consistem na recuperação de um BD em função de um instante, ou seja, elas retornam o histórico de uma ou várias informações dos estados do BD a partir de um ponto de tempo t. As consultas *walk-through* retornam a informação dos estados do BD, em diversos pontos ou intervalos de tempo, aplicando um predicado temporal sobre os mesmos. As duas consultas podem ser aplicadas a informações históricos ou *rollback*, dependendo do tempo ser do tipo válido ou de transação.

Uma consulta temporal tem dois componentes: a seleção temporal e a projeção temporal. A **seleção temporal** retorna dados em função de um predicado temporal [Brayner94]. Dentro do contexto de uma consulta, ela representa uma condição lógica estabelecida sobre informações temporais associadas aos dados [Edelweiss94]. Nas consultas *as-of*, o predicado temporal é especificado através dos operadores temporais ***Begin, End e At*** sobre os instantes, enquanto que nas consultas *walk-through*, o predicado temporal é especificado através de operadores temporais sobre os intervalos de tempo, e os mais comuns são: *Before, After, Overlap, During* [Allen84], [Ariav86], [Gadia88], [Käfer90], [Snodgrass87].

Após a seleção temporal, a **projeção temporal** retorna valores de tempo, que são representados através de atributos especificos [Brayner94]. Ela representa a saída de uma consulta que envolve informações temporais associadas aos dados [Edelweiss94]. A projeção temporal não é aplicada às consultas *as-of*, já que esta consulta pré-determina um instante de tempo desejado na própria condição. Nas consultas *walk-through*, a projeção temporal é especificada através de **construtores temporais**. Os propostos na maioria das linguagens de consulta temporais [Gadia88], [Snodgrass87], são: *Last_Instant, Last_Interval, First_Instant, First_Interval, Adjacent.*

Em [Edelweiss94], são analisadas as possíveis combinações entre os componentes seleção e projeção temporal que podem surgir em uma consulta temporal, classificando as consultas de seleção como: seleção

sobre dados, seleção temporal e seleção mista, além de classificar as consultas de projeção como: projeção sobre dados, projeção temporal e projeção mista. Seleção ou projeção sobre dados atuam somente sobre dados atemporais, seleção ou projeção temporal atuam sobre informações temporais associadas aos dados e seleção ou projeção mista atuam tanto nos dados, quanto nas informações temporais associadas a eles.

Baseado nas possíveis combinações acima, apresentamos alguns exemplos de consultas temporais e logo em seguida uma tabela de critérios nos quais serão enquadrados estes exemplos. A tabela não considera a combinação seleção e projeção sobre dados, já que não envolve tempo, e a combinação seleção e projeção temporal não pode ser utilizada, pois deve haver algum dado envolvido [Edelweiss94].

1. *Quais os empregados que foram admitidos em 10/01/94?*
2. *A partir de quando o empregado chamado João possui um salário > xxx?*
3. *Qual foi o último salário de cada empregado antes de 10/01/94, e a partir de quando eles obtiveram esse salário?*
4. *Qual foi o último salário do empregado chamado João, antes de 10/01/94, e a partir de quando ele obteve esse salário?*
5. *Qual o histórico salarial dos empregados da equipe do projeto xxx?*
6. *Em que data o empregado João subiu de nível pela primeira vez?*
7. *Qual era salário do empregado João quando ele subiu de nível? Na mudança deste salário, ele manteve o mesmo nível?*

	Sel. sobre dados	Sel. Temporal	Sel. Mista
Projeção sobre dados	////////////////////////////	**Exemplo 1**	**Exemplo 7**
Projeção Temporal	**Exemplo 2**	////////////////////////////	**Exemplo 6**
Projeção Mixta	**Exemplo 5**	**Exemplo 3**	**Exemplo 4**

Tabela 4.2. Classificação de Seleção/Projeção dos Exemplos de Consultas

A tabela 4.2 identifica, por exemplo, que no *Exemplo 3* é aplicado uma seleção temporal, pois a condição da consulta envolve somente o tempo ("*antes de 10/01/94*") e é aplicado uma projeção mista, pois envolve a recuperação tanto de valores de dados quanto o tempo associados a estes valores ("*Qual foi o último salário" e "a partir de quando"*). No *exemplo 7*, apesar de não haver um tempo explícito na consulta, a frase *"quando ele subiu de nível"* representa uma informação temporal, e por isso, a seleção é mixta.

Baseado nas diferentes possibilidades em especificar uma consulta temporal, em ConTOM, qualquer tipo de consulta temporal que seja feita, será composta por uma combinação de uma ou várias operações dos dois tipos de consulta já definidos (*as-of* e *walk-through)* para a seleção temporal, seguidos da projeção temporal. Portanto, vamos considerar os OGC's temporais similares às operações de seleção (OGC's *as-of e walk through*) e projeção temporais (OGC *projeção temporal*), a nível semântico. Os OGC's *as-of e*

walk-through são independentes, enquanto que o OGC *projeção temporal* se aplica somente sobre o resultado do OGC *walk-through*. O tempo permitido na especificação de uma consulta temporal em ConTOM é o tempo **válido** (o tempo de transação não é considerado). A seleção e projeção sobre dados não serão consideradas, pois não envolvem tempo e já são representadas nos OGC's básicos (*seleção* e *projeção)*. Os OGC's temporais serão descritos informalmente acompanhados de exemplos ilustrativos, como veremos a seguir.

4.2.3.4. OGC *As-Of*

Consiste na seleção de classes ou relacionamentos temporais, em função de um predicado temporal expresso da seguinte forma: *Operando1 OpTemp Operando2. Operando1* pode ser uma classe, um relacionamento ou um módulo que foi selecionado pelo usuário. *OpTemp* refere-se ao operadores temporais *Begin, End e At* já citados anteriormente. *Operando2* refere-se a um instante, que pode ser uma constante digitada pelo usuário, ou uma classe cujo domínio seja um intervalo de tempo. Uma condição com *Begin* considera dados cujo intervalo temporal tem *Operando2* como seu valor *início*, *End* considera dados em que o valor do instante fornecido seja igual ao seu valor *fim* e *At* considera dados em que o valor do instante fornecido esteja entre os seus dois valores *início* e *fim*, ou seja, um intervalo que satisfaça a condição: *início* <= ***instante*** <= *fim.*

O resultado será de dois tipos: se o OGC *as-of* é aplicado a uma classe temporal, o resultado será a mesma classe, não temporal, contendo somente as instâncias da classe que são válidas naquele instante. Se aplicado a um relacionamento temporal de uma classe *c1* a uma classe *c2*, o resultado será o mesmo relacionamento, não temporal, que foi válido, pelo menos, no instante *t* da consulta *as-of.*

Para uma melhor compreensão de uma consulta *as-of*, vamos considerar o *exemplo 1: "Quais os empregados admitidos em 10/01/94?"*. A partir do módulo de consultas da figura 4.21a, aplica-se o OGC *as-of* sobre a classe *empregado* (em destaque), e introduz-se o predicado "*Begin 10/01/94*" (a maneira de formular o predicado será visto no capítulo VI). O resultado será um novo módulo ilustrado na figura 4.21b:

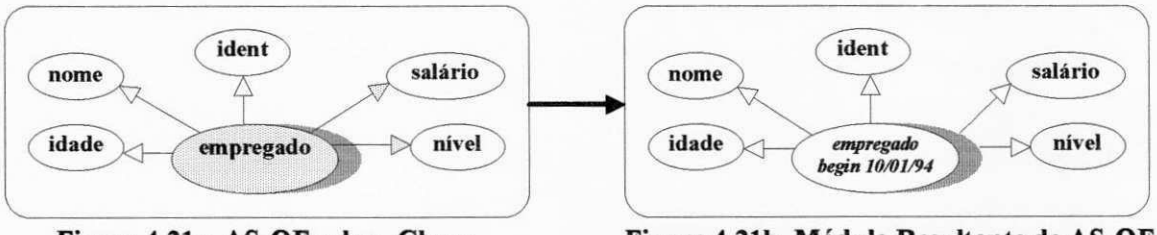


Figura 4.21a. AS-OF sobre Classe

Figura 4.21b. Módulo Resultante de AS-OF

Vamos introduzir um outro exemplo aplicando *as-of* sobre um relacionamento temporal: "*Quais eram os salários dos empregados em 10/01/94?*". Considerando o módulo de consultas da figura 4.22a, aplica-se o OGC *as-of* sobre o relacionamento temporal *tem-salário* entre as classes *empregado* e *salário*, com o predicado *At(10/01/94)*. O resultado será um novo módulo ilustrado na figura 4.22b (note que o relacionamento *tem-salário* deixa de ser temporal):

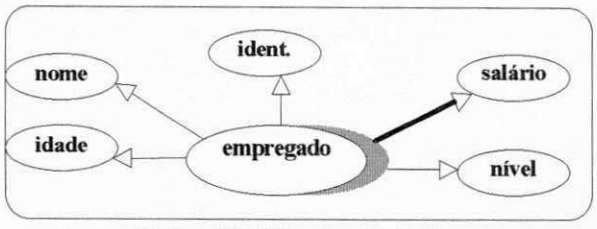


Figura 4.22a. AS-OF sobre Relacionamento

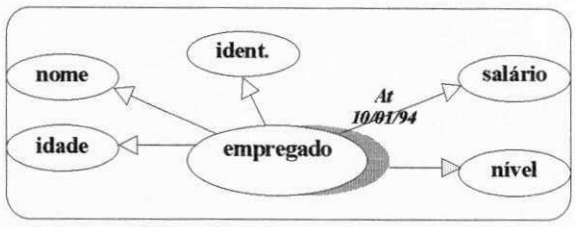


Figura 4.22b. Módulo Resultante de AS-OF

4.2.3.5. OGC *Walk-Through*

Consiste na pesquisa do histórico de objetos ou relacionamentos temporais, em função de um predicado temporal expresso da seguinte forma: *Operando1 OpTemp Operando2*, onde *Operando1* e *Operando2* são os mesmos da consulta *as-of* e *OpTemp* refere-se ao operadores temporais *Before, After, During* já citados anteriormente (na utilização do operador *During*, a entrada do *Operando2* será um intervalo de tempo). O resultado de um predicado especificado com os operadores acima serão os dados válidos no tempo especificado. Quando se deseja **selecionar** todo o histórico de uma informação, utiliza-se somente um operador especial, denominado *ever*.

Se o OGC *walk-through* é aplicado a uma classe temporal com os operadores temporais, o resultado será uma nova classe não temporal, obtida da agregação da classe objeto da consulta com uma classe de intervalos temporais que satisfaz a consulta. Se o OGC *walk-through* é aplicado a um relacionamento temporal *rel* de uma classe *c1* a uma classe *c2*, o resultado será uma classe agregada composta pelas próprias classes *c1* e *c2*, e uma classe de intervalos de tempo restrita ao(s) intervalo(s) especificado, identificando os relacionamentos que foram válidos durante este intervalo.

Entre a aplicação do *walk-through* e o resultado, existe uma etapa intermediária, onde o relacionamento temporal entre as classes c1 e c2 equivale a uma classe temporal agregada, cujos componentes são as próprias classes c1 e c2, se considerarmos que um relacionamento entre duas classes pode ser modelado como uma classe agregada entre elas. Contudo, esta etapa é transparente para o usuário, como ilustra a figura 4.23 (a etapa intermediária está no centro). A segunda etapa equivale à aplicação de um *walk-through* à classe temporal agregada, gerada na primeira etapa. Desta forma, aplicar um *walk-through* sobre um relacionamento temporal é o mesmo que aplicar sobre este relacionamento modelado como uma classe agregada temporal.

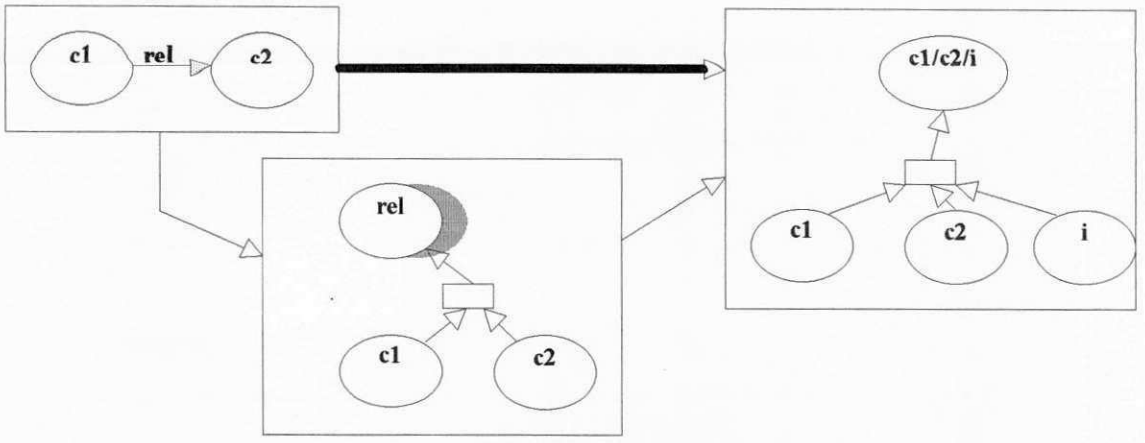


Figura 4.23. Aplicação de umWalk-Through sobre um Rel. Temporal

Para uma melhor compreensão de uma consulta *walk-through*, vamos considerar o seguinte exemplo: "*Quais foram os salários dos empregados em 1994?*". Vamos considerar o módulo de consultas ilustrado na figura 4.24a. A partir deste módulo, aplica-se o OGC *walk-through* sobre o relacionamento temporal *tem-salário* (em destaque na figura) e introduz-se o predicado "*During 01/01/94 - 31/12/94*". O resultado será um novo módulo ilustrado na figura 4.24b.

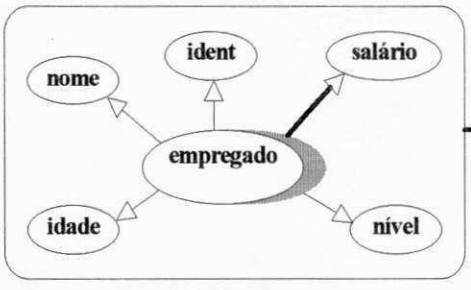


Figura 4.24a. Walk-Through em Relac.

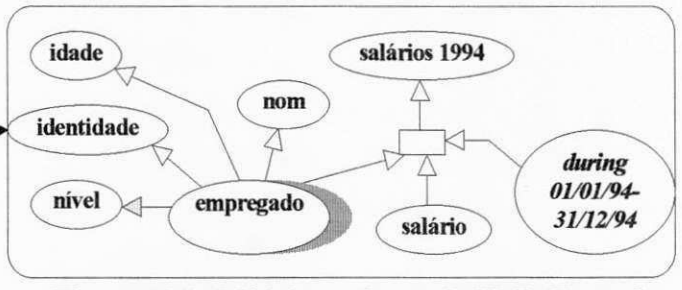


Figura 4.24b. Módulo resultante de Walk-Through

Vamos considerar um exemplo utilizando o operador *ever*, o *exemplo5*: "*Qual o histórico salarial de todos os empregados?*". Neste exemplo, aplica-se o OGC *walk-through* sobre o relacionamento temporal *tem-salário* no módulo de consultas da figura 4.25a, juntamente com o operador *ever* e será gerado um novo módulo da figura 4.25b.

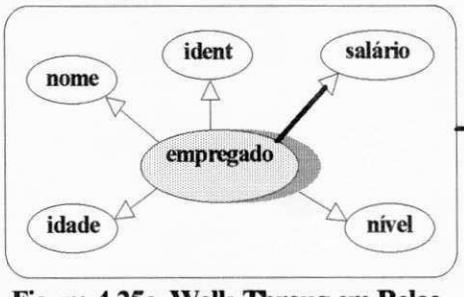


Figura 4.25a. Walk-Throug em Relac.

Figura 4.25b. Módulo resultante de Walk-Through

Vamos introduzir um outro exemplo aplicando *walk-through* sobre uma classe temporal: *"Quais os empregados após 10/01/94?"*. Considerando o módulo de consultas da figura 4.26a, aplica-se o OGC *walk-through* sobre a classe *empregado* com o predicado *"After 10/01/94"*. O resultado será um novo módulo contendo uma classe agregada temporal, ilustrado na figura 4.26b:

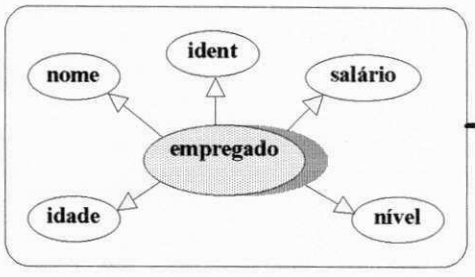


Figura 4.26a. Walk-Through em Classe

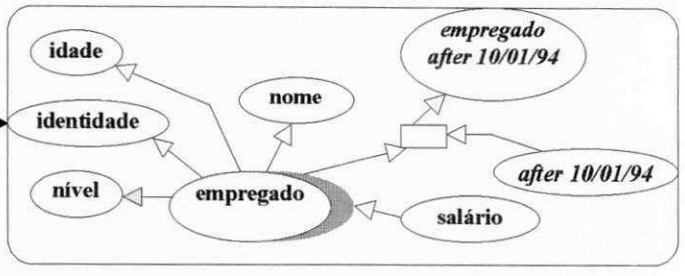


Figura 4.26b. Módulo resultante de Walk-Through

4.2.3.6. OGC *Projeção Temporal*

Consiste na seleção sobre a classe componente referente ao(s) intervalo(s), da classe agregada resultante da aplicação de um OGC *walk-through*, em função dos seguintes construtores temporais: *Last_Instant, Last_Interval, First_Instant, First_Interval,* já citados anteriormente. Uma projeção temporal, especificada com estes operadores, retorna uma subclasse da classe agregada, contendo as instâncias associadas ao(s) intervalo(s) selecionado(s) pelo predicado temporal de um *walk-through*. Quando não se deseja especificar os operadores anteriores, recuperando todo o histórico de uma informação, utiliza-se novamente o operador *ever*.

Para uma melhor compreensão de uma projeção temporal, vamos considerar o exemplo "*Quais foram os salários dos empregados em 1994? A partir de quando cada empregado obteve o último salário de 1994?* Vamos considerar o módulo ilustrado na figura 4.27b, resultante da aplicação de um *walk-through*. A partir deste módulo, aplica-se o OGC *projeção temporal* sobre a classe componente *during 01/01/94-31/12/94* juntamente com o construtor *Last_Interval*, já que deseja-se o último salário de cada empregado. Contudo, o usuário só deseja saber o início deste intervalo. Desta forma, aplica-se novamente a *projeção temporal* sobre a classe componente que representa o intervalo, juntamente com o operador *First_Instant*, gerando o módulo ilustrado na figura 4.27c.

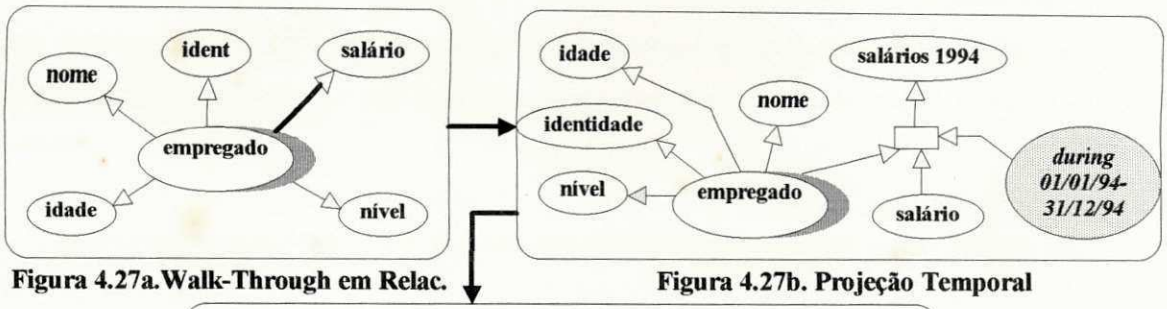


Figura 4.27a.Walk-Through em Relac. **Figura 4.27b. Projeção Temporal**

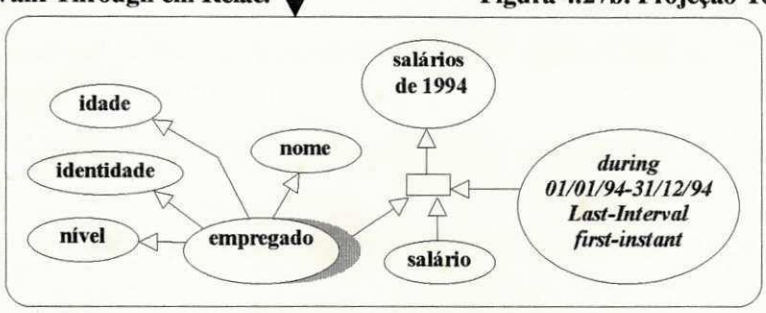


Figura 4.27c. Módulo resultante de Projeção Temporal

Vamos considerar outro exemplo similar, o *exemplo 6*, *"A partir de que período cada empregado subiu de nível pela primeira vez?"*. Vamos considerar que todo empregado entra com um nível inicial. A partir do módulo de consultas da figura 4.28a, o usuário aplica um *walk-through*, com o operador *ever*, sobre o relacionamento temporal *tem-nível* entre as classes *empregado* e *nível*, gerando o módulo da figura 4.28b. A partir deste módulo, aplica-se o OGC *projeção temporal* sobre a classe componente *ever*, juntamente com o construtor *First_Interval,* de forma a recuperar a validade dos primeiros níveis de cada empregado. Como no exemplo anterior, aplica-se novamente a *projeção temporal* sobre a nova subclasse juntamente com o operador *Last_Instant,*de forma a recuperar o instante em que ele subiu de nível pela primeira vez, considerando que o final do 1o. intervalo corresponde ao início do 2o. O resultado é ilustrado na figura 4.28c:

Figura 4.28a. Walk-Through em Relacionamento

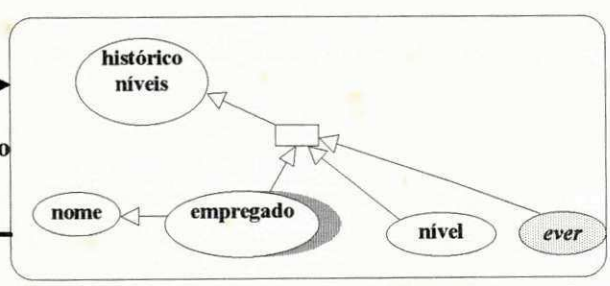


Figura 4.28b. Projeção Temporal

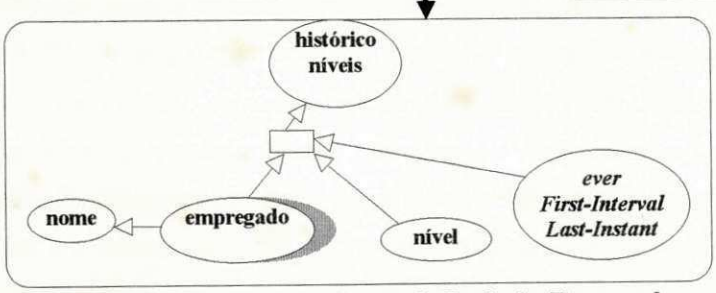


Figura 4.28c. Módulo Resultante da Projeção Temporal

Para finalizar, vamos apresentar exemplos que combinam os dois tipos de consulta (*as-of e walk-through*), cada exemplo tem ao seu lado, (em parênteses) a ordem de aplicação de um *as-of* ou *walk-through*:

"Quais eram os salários dos empregados ***em*** *10/01/94?* ***na*** *mudança desse salário, quais eram os seus níveis?" (as-of, as-of).*

"Quais eram os salários dos empregados em 10/01/94?, ***após*** *a mudança desse salário, quais eram os seus níveis?" (as-of, walk-through).*

"Quais eram os salários dos empregados ***quando*** *eles subiram de nível pela primeira vez? (walk-through, as-of).*

"Quais eram os salários ***após*** *eles subirem de nível pela primeira vez? (walk-through, walk-through).*

"Quais os candidatos que foram admitidos ***quando*** *os empregados, do extinto departamento xxx, foram demitidos ? (as-of, as-of).*

"Quais os candidatos que foram admitidos ***após*** *os empregados, do extinto departamento xxx, terem sido demitidos ? (as-of, walk-through).*

"Quais os candidatos que foram admitidos ***no*** *primeiro período da demissão dos empregados, do hoje extinto departamento xxx, após 10/01/94?" (walk-through, as-of).*

"Quais os candidatos que foram admitidos após o primeiro período da demissão dos empregados, do hoje extinto departamento xxx, após 10/01/94?" (walk-through, walk-through).

"Qual o primeiro salário de cada empregado que foi admitido ***em*** *10/01/94?" (as-of, as-of).*

"Qual o histórico salarial dos empregados que foram admitidos ***em*** *10/01/94?"* ***(****as-of, walk-through****).***

"Qual o primeiro salário de cada empregado que foi admitido ***após*** *10/01/94?" (walk-through, as-of).*

"Qual o histórico salarial de cada empregado que foi admitido ***após*** *10/01/94?"* ***(****walk-through, walk-through****).***

A título de ilustração, vamos considerar o 3º exemplo: *"Quais eram os salários dos empregados* ***quando*** *eles subiram de nível pela primeira vez?".* A partir do módulo de consultas da figura 4.29a, aplica-se um *walk-through*, com o operador *ever*, sobre o relacionamento temporal *tem-nível*, gerando o módulo da figura 4.29b. A partir deste módulo, o usuário aplica uma *projeção temporal*, com os operadores *first-interval* e *last-instant*, gerando o módulo da figura 4.29c. A partir deste módulo, aplica-se um *as-of* sobre o relacionamento *tem-salário* cujo *Operador2* é *At(t)*, em que *t* é obtido do componente *last-instant* do agregado *histórico-níveis*, cujo valor refere-se à primeira mudança de nível de cada empregado. O resultado é o módulo da figura 4.29d. Após a geração deste módulo, as classes de interesse são projetadas normalmente.

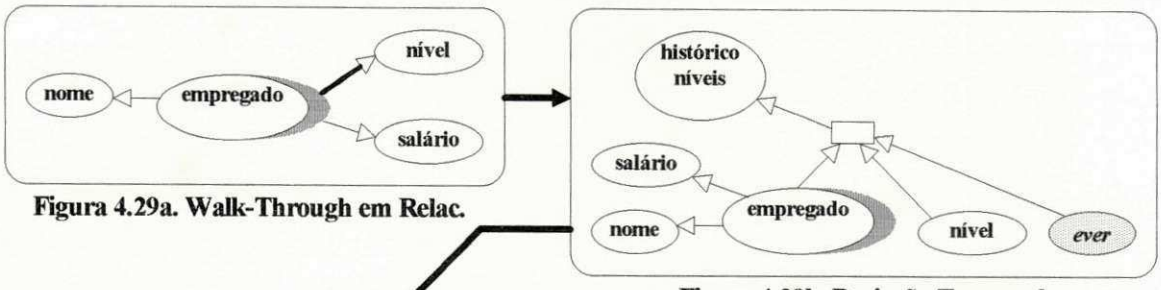


Figura 4.29a. Walk-Through em Relac.

Figura 4.29b. Projeção Temporal

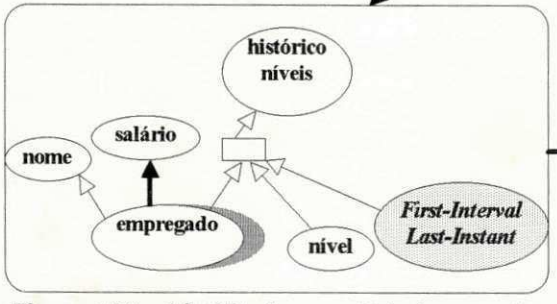


Figura 4.29c. AS-OF sobre um Relacionamento

Figura 4.29d. Módulo Resultante de AS-OF

4.3. Semântica Formal dos OGC's

O formalismo apresentado nesta seção é baseado no trabalho de *Tiziana Catarci* [Catarci93], que fornece uma definição formal de um conjunto mínimo de **primitivas gráficas** que são fundamentais às linguagens visuais de consulta (LVC). Seu trabalho adota uma abordagem de unificação de um modelo de dados e sua representação gráfica, denominando o modelo de *graph model* e aplicando diretamente sobre ele operações gráficas (ex: seleção de um nodo) com sua própria semântica. *Catarci* comprova que um *graph model* e as primitivas gráficas fornecem os princípios gerais para o desenvolvimento de interfaces gráficas para BD's heterogêneos, sendo possível utilizá-los como constituintes básicos de representações visuais mais complexas e LVC's, fornecendo uma semântica independente do modelo sob a qual ela se fundamenta. Portanto, como a sua abordagem é similar ao nosso trabalho proposto, utilizaremos o seu formalismo, adaptando-o à definição do ECG e aos OGC's básicos e temporais.

4.3.1. Formalismo de um ECG

Considerando que todo o processo de especificação de uma consulta é a aplicação de OGC's sobre um ECG, é necessário formalizarmos o ECG, baseado no formalismo de um *graph model* [Catarci93]. Um *graph model* é definido como uma tripla $<g,r,i>$, onde g é a parte gráfica e corresponde a um ECG; r corresponde a um conjunto de restrições e i corresponde à interpretação do modelo. g *e* r representam a parte intensional do modelo enquanto que i representa a parte extensional, isto é, as instâncias do BD.

O ECG é definido como uma tupla $<N, L, H, D, Fd, Fs>$, onde:

$N = Nn \cup Np \cup Nt$ é o conjunto de **nodos**. *Nn* são nodos que representam as classes não primitivas, *Np* são nodos que representam as classes primitivas e *Nt* são nodos que representam as classes temporais;

$L = Ln \cup Lh \cup Lt$ é o conjunto de **ligações**, representando o relacionamento entre classes. $Ln \subseteq N \times N$ é o conjunto de relacionamentos entre nodos, $Lh \subseteq N \times H \cup H \times N$ são os relacionamentos entre nodos e nodos hierárquicos e $Lt \subseteq N \times N$ são os relacionamentos temporais;

$H = Hg \cup Ha \cup Hp$ é o conjunto dos **nodos hierárquicos** e representam as conexões hierárquicas de generalizações, agregações e agrupamentos, respectivamente.

D é um conjunto de **denominações**;

Fd é uma **função** biunívoca de $N \cup Ln \cup Lt \cup Hg$ para *D*, que associa uma denominação a cada nodo ou ligação;

Fs é uma **função** que caracteriza a seleção de elementos do ECG, mapeando cada nodo ou ligação a um valor {*não selecionado*, *selecionado*};

Durante o decorrer desta seção, serão utilizadas as seguintes notações:

AD(n) são os nodos adjacentes a um dado nodo ***n***.

AD(l) são os nodos adjacentes $\{n_1, n_2\}$ a uma ligação ***l***. Se $AD(l) = \{n_1, n_2\}$ então $n_1 \in AD(n_2)$ e $n_2 \in AD(n_1)$. Usaremos também a notação $n_1\ l\ n_2$.

$T = \{t\} \cup \{agora\}$ é o conjunto de instantes de tempo *t* e o valor especial ***agora***. Observamos que *T* é um conjunto totalmente ordenado, isto é, $\forall t_1, t_2 \in T\ (t_1 \leq t_2 \vee t_2 \leq t_1)$.

Int é o conjunto de intervalos de tempo, representado por pares $<t_1, t_2> \mid t_1, t_2 \in T$.

$O = Op \cup On \cup Ot$ é um conjunto de objetos. ***Op*** são os objetos primitivos, ***On*** são os objetos não primitivos e ***Ot*** são os objetos temporais.

$R = Rn \cup Rt$ é o conjunto de relacionamentos. ***Rn*** são os relacionamentos não temporais e ***Rt*** são os relacionamentos temporais. *R* pode ser caracterizado por $R \subseteq O \times O$ e $<o_1, o_2> \in R \Rightarrow \exists n_1, n_2 \in N, l \in L\ (n_1\ l\ n_2 \wedge o1 \in n_1 \wedge o_2 \in n_2)$.

$U = O \cup R$ é um universo definido como um conjunto de objetos e relacionamentos entre objetos.

Θ: $U \rightarrow$ *Int* é uma função que associa a cada objeto ou relacionamento um intervalo, denotado por $\Theta(x) = < begin(x), end(x)>$.

Se $t \in <begin(x), end(x)>$, $\boldsymbol{x_t}$ significa o estado do objeto ou relacionamento x no instante t.

O estado de um objeto em t significa os valores dos atributos e relacionamentos naquele instante. O estado de um relacionamento $o1\ l\ o2$ em t pode ser *true* ou *false*, dependendo se era válido ou não naquele instante. Definimos o estado de um relacinamento, a nível de nodo, como: $(n1\ l\ n2)_t = \{o1\ l\ o2 \mid (o1\ l\ o2) = true\}$.

Elementos não temporais de U são os que tem validade no presente, ou seja: Se $\boldsymbol{x \in Op \cup On \cup Rn}$, então $begin(x) = end(x) = \boldsymbol{agora}$.

Seja ECG = $<N, L, H, D, Fd, Fs>$. Uma **interpretação** para ECG é uma função $\boldsymbol{i: N \cup L \rightarrow 2^u}$, mapeando cada nodo $n \in N$ e $l \in L$ a um subconjunto de U, como segue:

Se $n \in Np$, então $i(n) \subseteq Op$;
Se $n \in Nn$, então $i(n) \subseteq On$;
Se $n \in Nt$, então $i(n) \subseteq Ot$;
Se $l = <n_1, n_2> \in Ln$, então $i(l)$ é um conjunto de pares da forma $<o_1, o_2> \in Rn$ tal que $o_1 \in i(n_1)$ e $o_2 \in i(n_2)$.
Se $l = <n_1, n_2> \in Lh$, então $i(l) = \varnothing$
Se $l = <n_1, n_2> \in Lt$, então $i(l)$ é um conjunto de pares da forma $<o_1, o_2> \in Rt$ tal que $o_1 \in i(n_1)$ e $o_2 \in i(n_2)$.
Se $n \in Nt$, a interpretação de n em um instante t é dado por $i(n_t) = \{ot \mid o \in i(n)\}$
Se $l \in Lt$, a interpretação de l em um instante t é dado por $i(l_t) = \{o_1\ l\ o_2 \mid \exists n_1, n_2\ (o_1 \in n_1 \wedge o_2 \in n_2 \wedge n1\ l\ n2 \wedge (o_1\ l\ o_2)_t = true\}$.

As restrições são especificadas através de uma **Linguagem de Restrição** contendo as características do modelo TOM que são relevantes para o formalismo dos OGC's e que possam ser graficamente representadas em um ECG, descritas a seguir:

n_1 ***é-um*** n_2; onde $n_1, n_2 \in Nn$ e é satisfeita se $i(n_1) \subseteq i(n_2)$ e $\exists h, l_1, l_2\ (h \in Hg, l_1, l_2 \in Lh \mid n_1\ \boldsymbol{l_1}\ h \wedge h\ \boldsymbol{l_2}\ n_2)$;

$\{n_1,...,n_k\}$ ***é-parte*** n; onde $n_1,...,n_k \in N$ e $n \in Nn$ e é satisfeita se $i(n) \subseteq i(n_1)$x...x $i(n_k)$ e $\exists h, l_1, l_2\ (h \in Ha, l_1, l_2 \in Lh \mid \{n_1,...,n_k\}\ \boldsymbol{l_1}\ h \wedge h\ \boldsymbol{l_2}\ n)$;

n_1 ***é-elemento*** n_2; onde $n_1, n_2 \in Nn$ e é satisfeita se $i(n_2) \subseteq 2^{n}1$ e $\exists h, l_1, l_2 (h \in Hp, l_1, l_2 \in Lh \mid n_1\ \boldsymbol{l_1}\ h \wedge h\ \boldsymbol{l_2}\ n_2)$;

A cada $l = <n_1, n_2> \in Ln \cup Lt$ estão associadas duas cardinalidades ***min(l)*** e ***max(l)***, chamadas restrições de cardinalidade. Dado um $o_1 \in i(n_1)$, denotamos por $o_1 l$ os elementos de n_2 ligados a o_1, ou seja: $o_1 l = \{o_2 \mid o_1\ l\ o_2\}$. A seguinte regra de cardinalidade faz parte das restrições do *graph model*: $\forall\ l \in Ln \cup Lt\ (l = <nl, n_2> \wedge o_1 \in n_1 \Rightarrow min(l) \leq \# o_1, l \leq max(l))$.

A partir das restrições que foram definidas, vamos adicionar outras notações que serão utilizadas no decorrer desta seção:

AD'(n) é o conjunto definido como: se $n \in N$, então $AD'(n) = AD(n) \cup {}_iAD(n_i)$ onde $n_i \in Nn$ e n ***é-um**** n_i. Em outras palavras, o conjunto $AD'(n)$ contém os nodos adjacentes a n e os nodos adjacentes às suas superclasses.

AD'{l) é o conjunto definido como: se $l \in L$ e $AD(l) = \{n_1, n_2\}$, então $AD'(l) = AD(l) \cup \{n \in Nc \mid n$ ***é-um*** $n_1 \vee n$ ***é-um*** $n_2\}$. Em outras palavras, o conjunto $AD'(l)$ contém os seus nodos adjacentes mais os descendentes de tais nodos.

4.3.2. Formalismo dos OGC's

Nesta seção apresentaremos o formalismo dos OGC's básicos e temporais. Os OGC's serão constituídos de **primitivas gráficas** mais simples que correspondem às diferentes interações do usuário, na utilização de um OGC. Seja, por exemplo M = $<ECG, r, i>$ um módulo inicial sobre o qual serão aplicados as primitivas gráficas do OGC *seleção*; M' = $<ECG', r', i'>$, é o módulo intermediário após a aplicação das primitivas gráficas e $M^r = <ECG', r', i'>$ o módulo resultante após a aplicação de *seleção*. O OGC *seleção* pode ser aplicado mais de uma vez, gerando os módulos resultantes provisórios M_1^r, M_2^r,.....,. O M^r final será a união dos módulos provisórios. Ao aplicar um outro OGC, o Mr gerado se torna novamente M para a aplicação deste OGC. Este processo continua até o término da especificação da consulta. O último Mr corresponde ao resultado da consulta (ER).

4.3.2.1. OGC *Seleção*

Geralmente, deseja-se selecionar um caminho entre classes não primitivas e alguns atributos destas classes. Definimos um **caminho** em um esquema conceitual gráfico ECG, como uma sequência de classes não primitivas conectadas $n_1\ l_1\ n_2, n_2\ l_2\ n_3,..., n_k\ l_k\ n_{k+1}$ que denotamos como $p = <n_1, l_1, n_2, l_2, n_3,...,$

n_k, l_k, n_{k+1}> ou, se os elementos intermediários não são de interesse, p = <n_1,..,n_{k+1}>. Dado um caminho *p*, a interpretação do caminho *i(p)* é o conjunto de todos os objetos conectados o_1 l_1 o_2, o_2 l_2 o_3,..., o_k l_k o_{k+1}, e dizemos que $o \in i(p)$ *se* $o \in \{o_1,..,o_{k+1}\}$.

⇒ Seja $p = <n_1, l_1, n_2, l_2, n_3,..., n_k, l_k, n_{k+1}>$ um caminho em um módulo M, and $n_p = \{n_{p1}, ..,n_{ps}\}$ um conjunto de nodos primitivos conectados aos nodos do caminho, a **seleção do caminho p com atributos n_p em M** é uma função σ(M, *p*, n_p) = M'. M' é igual a M exceto para cada elemento x do caminho e cada elemento y dos atributos:

Fs'(x) = selecionado; e *Fs'(y) = selecionado;*

A seleção de uma classe única significa selecionar um caminho de tamanho 1. Cada parte do caminho onde não existe ambiguidade, apenas os extremos devem ser declarados. Por exemplo, se existe um único caminho de n_1 to n_2, podemos escrever σ(M, <n1,n2>, n_p).

Após o término do processo de seleção, o $M^r = <ECG^r, r^r, i^r>$ é um subesquema de M tal que:

$N^r = \forall n \in N' | Fs'(n) = selecionado$;

$L^r = \forall l \in L' | Fs'(l) = selecionado$;

$H^r = \forall h \in H', \exists l \in L, \exists n,m \in N \mid n\ l\ h \wedge h\ l\ m \wedge Fs'(n) = selecionado \wedge Fs'(m) = selecionado$; **Isto é necessário para herança;

$Fs^r = \forall n, l, n \in N \wedge l \in L \mid Fs^r(n) = não\ selecionado \wedge Fs^r(l) = não\ selecionado$;

$r^r = r' \cup \forall\ n_1, n_2, l^r\ (n_1, n_2 \in N, l^r \in L^r \Rightarrow min(l^r) = sup\{min(l'),\ 1\})$ **objetos não relacionados desaparecem;

$\forall n \in p, i^r(n) = \{o \in i'(n) \mid o \in i(p)$

Se o processo de seleção é aplicado mais de uma vez, teremos vários módulos resultantes M_1^r, M_2^r,....., M_k^r. Neste caso, um passo adicional irá gerar um módulo integrado M^r como união dos módulos intermediários, porém, classes que aparecem duas ou mais vezes são duplicadas, desde que as instanciações são diferentes, devido às diferentes interpretações. Teremos então:

$M^r = \cup M_i^r$, exceto que: para todo par M_i^r e M_1^r:

Se $n \in N_i^r$, então haverá dois nodos *n'* e *n''* em N^r tal que:

$Fd(n') = Fd_i(n)$ *conc "(i)"* e

$Fd(n'') = Fd_j(n)$ *conc "(j)"* e

$i(n') = i_i(n)$ *e* $i(n'') = i_i(n)$.

4.3.2.2. OGC *Especialização*

O processo de especialização é constituído das seguintes primitivas:

⇒ Seja $n \in Nn$, e P um predicado sobre um atributo $p \in AD(n)$, a **especialização de um nodo *n*** em M: ε (M, n, P) = M' tal que:

M' = M, exceto para: σ_V (M,n, p).

⇒ Seja $n, m \in Nn$ e $P \in \{\cap, -\}$ um predicado, a **especialização múltipla dos nodos *n e m*** em M: εm (M, n, m, P) = M' tal que:

M' = M, exceto para:

a) $Fs'(n) = selecionado \wedge Fs'(m) = selecionado$;

b) $r' = r \cup (\exists n' \in Nn \mid n$ ***é-um*** n' e m ***é-um*** $n')$;

Após o término do processo de especialização, o $M^r = <ECG^r, r^r, i^r>$ é igual a M', exceto para:

$Nn^r = N'n' \cup \{s\}$;

$Lh^r = Lh' \cup \{<n, h>, <h,s>\}$;

$H^r = H' \cup \{h \mid h \in Hg\}$;

$D^r = D' \cup \{Fd'(n)$ *conc* "P" $\vee$ $Fd'(n)$ *conc* "P" *conc* $Fd'(m)\} \cup \{$"P"$\}$;

$Fd^r = Fd' \cup \{Fd(s) = Fd'(n)$ *conc* "P" $\vee$ $Fd'(n)$ *conc* "P" *conc* $Fd'(m)$, $Fd(h) =$ "P"$\}$;

$Fs^r = \forall n \in N \mid Fs^r(n) =$ *nao selecionado*;

A restrição r^r e a interpretação i^r em M^r é diferente para ε e εm. Em ε, temos a seguinte restrição e interpretação:

$r^r = r' \cup \{s$ ***é-um*** $n\}$.

$i^r = i' \cup \{o \in i^r(n) \mid P(o)\}$.

Em εm, temos a seguinte restrição e interpretação:

se $P = \cap \Rightarrow r^r = r' \cup \{s$ ***é-um*** $n,\ s$ ***é-um*** $m\}$;

se $P = -\ \Rightarrow r^r = r' \cup \{s$ ***é-um*** $n\}$.

se $P = \cap \Rightarrow i^r = i' \cup \{i^r(s) = (i^r(n) \cap i^r(m))\}$;

se P = - $\Rightarrow i^r = i' \cup \{i^r(s) = (i^r(n) - i^r(m))\}$;

4.3.2.3. OGC *Generalização*

⇒ Seja $n,m \in Nn$, a **generalização dos nodos *n e m*** em M: γ (M, *n, m,*) = M' tal que:

M' = M exceto para: $Fs'(n) = selecionado \wedge Fs'(m) = selecionado$;

Após o término do processo de generalização, o $M^r = <ECG^r, r^r, i^r>$ é igual a M', exceto para:

$Nn^r = Nn' \cup \{s\}$;

$Lh^r = Lh' \cup \{<n, h>, <m, h>, <h,s>\}$;

$H^r = H' \cup \{h \mid h \in Hg\}$;

$D^r = D' \cup \{Fd'(n)\ conc\ "\cup"\ conc\ Fd'(m)\} \cup \{"\cup"\}$;

$Fd^r = Fd' \cup \{Fd(s) = Fd'(n)\ conc\ "\cup"\ conc\ Fd'(m), Fd(h) = "\cup"\}$;

$Fs^r = \forall n \in N \mid Fs^r(n) = nao\ selecionado$;

$r^r = r' \cup \{n$ ***é-um*** s, m ***é-um*** $s\}$;

$i^r = i' \cup \{i^r(s) = (i^r(n) \cup i^r(m))\}$.

4.3.2.4. OGC *Agregação*

⇒ Seja $p_1,..., p_n \in Np$, a **agregação dos nodos $p_1,...,p_n$** em M: α (M, $p_1,...,p_n$) = M' tal que:

M' = M, exceto para: $Fs'(p_1) = selecionado \wedge ... \wedge Fs'(p_n) = selecionado$;

Após o término do processo de agregação, o $M^r = <ECG^r, r^r, i^r>$ é igual a M', exceto para:

$Nn^r = Nn' \cup \{s\}$;

$Lh^r = Lh' \cup \{<p_1, h>, ..., <p_n, h>, <h,s>\}$;

$H^r = H' \cup \{h \mid h \in Ha\}$;

$D^r = D' \cup \{$"Resultado"$\}$;

$Fd^r = Fd' \cup \{Fd(s) =$ "Resultado"$\}$;

$Fs^r = \forall n \in N \mid Fs^r(n) = nao\ selecionado$;

$r^r = r' \cup \{p_1,..., p_n$ ***é-parte*** $s\}$;

$i^r = i' \cup \{i^r(s) \subseteq i^r(p_1)$ x...x $i^r(p_n)\}$.

4.3.2.5. OGC *Agrupamento*

⇒ Seja $n, p \in Nn$, o **agrupamento do nodo *n* em relação ao nodo *p*** em M: ρ (M, *n, p*) = M' tal que:

M' = M, exceto para: $Fs'(n) = selecionado \wedge Fs'(m) = selecionado$;

Após o término do processo de agrupamento, o $M^r = <ECG^r, r^r, i^r>$ é igual a M', exceto para:

$Nn^r = Nn' \cup \{s\}$;

$Ln^r = Ln' \cup \{<s, p>\}$;

$Lh^r = Lh' \cup \{<n, h>, <h,s>\}$;

$H^r = H' \cup \{h \mid h \in Hp\}$;

$D^r = D' \cup \{Fd'(n)$ *conc* "por" *conc* $Fd'(m)\} \cup \{$"por"$\}$;

$Fd^r = Fd' \cup \{Fd(s) = Fd'(n)$ *conc* "por" *conc* $Fd'(m)$, $Fd(h) =$ "por", $Fd(<s, p>) = Fd(<n, p>)\}$;

$Fs^r = \forall n \in N \mid Fs^r(n) =$ *não selecionado*;

$r^r = r' \cup \{n$ ***é-elemento*** $s\}$;

$i^r = i' \cup \{i^r(s) \subseteq 2^{ir(n)}\}$.

4.3.2.6. OGC *As-Of*

O OGC *as-of* possui os seguintes parâmetros: ***x*** é um nodo ou uma ligação; ***P*** é um predicado temporal da forma *begin(t)*, *end(t)* ou *at(t)*, ***t*** é um instante de *T*.

⇒ Seja $n \in Nt$, e *P* um predicado temporal, um ***as-of* sobre o nodo *n* em relação a *P*** em M: ϕn (M, *n*, *P*) = M' tal que M' = M, exceto para: $Fs'(n) = selecionado$;

Após o término do *as-of* sobre nodo, o $M^r = <ECG^r, r^r, i^r>$ é igual a M', exceto para:

$Nn^r = Nn' \cup \{s\}$;

$Nt^r = Nt' - \{n\}$;

$D^r = D' \cup \{Fd'(n)$ *conc* "P"$\}$;

$Fd^r = Fd' \cup \{Fd^r(s) = Fd'(n)\}$;

$Fs^r = \forall n \in N \mid Fs^r(n) = nao\ selecionado$;

$r^r = r' \cup \{s = n_t\}$;

Se $P = begin(t)$, $i^r = i' \cup \forall o\{o \in i^r(s) \Leftrightarrow \exists o' \in i'(n)\ (begin(o') = t \wedge o = o'_{begin(o')})\}$;

Se $P = end(t)$, $i^r = i' \cup \forall o\{o \in i^r(s) \Leftrightarrow \exists o' \in i'(n)\ (end(o') = t \wedge o = o'_{end(o')})\}$;

Se $P = at(t)$, $i^r = i' \cup \forall o\{o \in i^r(s) \Leftrightarrow \exists o' \in i'(n)\ (o = o'_t)\}$;

⇒ Seja $l \in Lt$, $t \in$ T e P(t) um predicado temporal, um ***as-of* sobre a ligação *l* em relação a P** em M: ϕl (M, *l*, *P(t)*) = M' tal que M' = M, exceto para: $Fs'(l) = selecionado$;

Após o término do *as-of* sobre a ligação, o $M^r = <ECG^r, r^r, i^r>$ é igual a M', exceto para:

$Ln^r = Ln' \cup \{s\}$;

$Lt^r = Lt' - \{l\}$;

$D^r = D' \cup \{Fd'(l)$ *conc* "P"$\}$;

$Fd^r = Fd' \cup \{Fd^r(s) = Fd'(l)\}$;

$Fs^r = \forall l \in L \mid Fs^r(l) = nao\ selecionado$;

$r^r = r' \cup \{s = l_t\}$;

Se $P = begin(t)$, $i^r = i' \cup \{o_1\ s\ o_2 \in i^r(s) \Leftrightarrow o_1\ l\ o_2 \in i'(l) \wedge (o_1\ l\ o_2)_{begin(o1\ l\ o2)} = true\}$;

Se P = $end(t)$, $i^r = i' \cup \{o_1\ s\ o_2 \in i^r(s) \Leftrightarrow o_1\ l\ o_2 \in$ i'$(l) \wedge (o_1\ l$ o2$)_{end(o1\ l\ o2)} = true\}$;

Se $P = begin(t)$, $i^r = i' \cup \{o_1\ s\ o_2 \in i^r(s) \Leftrightarrow o_1\ l\ o_2 \in$ i'$(l) \wedge (o_1\ l\ o_2)_t = true\}$;

4.3.2.7. OGC *Walk-Through*

O OGC *walk-through* possui os seguintes parâmetros: ***x*** é um nodo ou uma ligação; ***P*** é um predicado temporal da forma *before(t), after(t), during(I),* ou *ever,* ***t*** é um instante de *T* e *I* é um intervalo de *Int.*

⇒ Seja $n \in Nt$, e *P* um predicado temporal, um ***walk-through*** **sobre o nodo *n* em relação a *P*** em M: ωn (M, *n, P)* = M' tal que M' = M, exceto para: *Fs'(n) = selecionado*;

Após o término do *walk-through* sobre nodo, o $M^r = <ECG^r, r^r, i^r>$ é igual a M', exceto para:

$Nn^r = Nn' \cup \{s, int\}$;

$Nt^r = Nt'$;

$H^r = H' \cup \{h\}$, com $h \in Ha$;

$Lh^r = Lh' \cup \{<n, h>, <int, h>, <h,s>\}$;

$D^r = D' \cup \{Fd'(n)\ conc\ "P"\} \cup \{"P"\}$;

$Fd^r = Fd' \cup \{Fd^r(s) = Fd'(n)\ conc\ "P", Fd^r(int) = "P"\}$;

$Fs^r = \forall n \in N \mid Fs^r(n) = nao\ selecionado$;

$r^r = r' \cup \{n$ ***é-parte*** s, int ***é-parte*** $s\}$;

Se $P = before(t)$, $i^r = i' \cup \{o \in i^r(s) \Leftrightarrow \exists o' \in i'(n)\ (o'$ *é-parte* $o \wedge \Theta(o') \cap before(t)$ *é-parte* $o)\}$;

Se $P = after(t)$, $i^r = i' \cup \{o \in i^r(s) \Leftrightarrow \exists o' \in i'(n)\ (o'$ *é-parte* $o \wedge \Theta(o') \cap after(t)$ *é-parte* $o)\}$;

Se $P =during(t)$, $i^r = i' \cup \forall o\{o \in i^r(s) \Leftrightarrow \exists o' \in i'(n)\ (o'$ *é-parte* $o \wedge \Theta(o') \cap I$ *é-parte* $o)\}$;

⇒ Seja $l = <n_1, n_2> \in$ Lt, e *P* um predicado temporal, um ***walk-through*** **sobre a ligação *l* em relação a *P*** em M: ωl (M, *l,, P)* = M' tal que M' = M, exceto para: *Fs'(l) = selecionado*;

Após o término do *walk-through* sobre ligação, o $M^r = <ECG^r, r^r, i^r>$ é igual a M', exceto para:

$Nn^r = Nn' \cup \{s, int\}$;

$Nt^r = Nt'$;

$H^r = H' \cup \{h\}$, com $h \in Ha$;

$Ln^r = Ln' - \{l\}$;

$Lh^r = Lh' \cup \{<n_1, h>, <n_2, h>, <int, h>, <h,s>\}$;

$D^r = D' \cup \{Fd'(n_1)\ conc\ Fd'(n_2)\ conc\ "P"\}$;

$Fd^r = Fd' \cup \{Fd^r(s) = Fd'(n_1)$ *conc* $Fd'(n_2)$ *conc* "P", $Fd^r(int)$ = "P"};

$Fs^r = \forall n \in N \mid Fs^r(n) =$ *nao selecionado*;

$r^r = r' \cup \{n1$ ***é-parte*** $s, n2$ ***é-parte*** s, int ***é-parte*** $s\}$;

Se $P = before(t)$, $i^r = i' \cup \{<o_1, o_2, I> \in i^r(s) \Leftrightarrow o_1 \, l \, o_2 \in i'(l) \wedge I = \Theta(o_1 \, l \, o_2) \cap before(t)\}$.

Se $P = after(t)$, $i^r = i' \cup \{<o_1, o_2, I> \in i^r(s) \Leftrightarrow o_1 \, l \, o_2 \in i'(l) \wedge I = \Theta(o_1 \, l \, o_2) \cap after(t)\}$.

Se $P = during(I_0)$, $i^r = i' \cup \{<o_1, o_2, I> \in i^r(s) \Leftrightarrow o_1 \, l \, o_2 \in i'(l) \wedge I = \Theta(o_1 \, l \, o_2) \cap I_0\}$.

Se $P = ever$, $i^r = i' \cup \{<o_1, o_2, I> \in i^r(s) \Leftrightarrow o_1 \, l \, o_2 \in i'(l) \wedge I = \Theta(o_1 \, l \, o_2)$.

4.3.2.8. OGC *Projeção Temporal*

O operador *Walk-through* gerou uma classe agregada *s* com um componente *int* que contém intervalos de tempo. A *Projeção Temporal* aplica-se sobre esta classe agregada para selecionar tempos específicos de interesse para a consulta.

⇒ Seja $n, int \in Nn$, as classes geradas por um operador *walk-through* e *P* um predicado temporal da forma: *first-instant(first-interval(int))*, *first-instant(last-interval(int))*, *last-instant(first-interval(int))*, ou *last-instant(last-interval(int)*, uma ***projeção temporal*** **sobre os nodos *n*** e ***int*** **com *P*** em M é uma função: $\pi(M, n, int, P) = M'$ tal que M' = M, exceto para: $Fs'(n) = Fs'(int) =$ *selecionado*.

Após o término da *projeção temporal* sobre o nodo, o $M^r = <ECG^r, r^r, i^r>$ é igual a M', exceto para:

$Nn^r = Nn' \cup \{int'\} - \{int\}$;

$D^r = D' \cup \{"P"\}$;

$Fd^r = Fd' \cup \{Fd^r(int') = "P"\}$;

$Fs^r = \forall n \in N \mid Fs^r(n) =$ *não selecionado*;

$r^r = r' \cup \{int'$ ***é-parte*** $n\}$;

Se $P =$ *first-instant(first-interval(int))*, $i^r = i' \cup \{t \in i^r(int') \Leftrightarrow \exists o' \in i'(n)$ (t = inf{*begin*(***tem-parte***(o', int)))

Se $P =$ *first-instant(last-interval(int))*, $i^r = i' \cup \{o \in i^r(int') \Leftrightarrow \exists o' \in i'(n)$ (t = sup{*begin*(***tem-parte***(o', int)))

Se $P =$ *last-instant (first-interval(int))*, $i^r = i' \cup \{o \in i^r(int') \Leftrightarrow \exists o' \in i'(n)$ (t = inf{*end*(***tem-parte***(o', int)))

Se $P =$ *last-instant(last-interval(int))*, $i^r = i' \cup \{o \in i^r(int') \Leftrightarrow \exists o' \in i'(n)$ (t = sup{*end*(***tem-parte***(o', int)))

Cada um dos operadores definidos acima é composto de dois passos: seleção e generalização. A aplicação de tal operador, implica na execução dos dois passos.

4.3.2.9. Exemplos utilizando o Formalismo

Mostraremos, nesta seção, como os operadores gráficos de consulta formais são aplicados para a realização de consultas concretas. Começaremos com uma consulta não temporal complexa:

1) Consulta não Temporal:

"Selecione projetos onde os membros da equipe são de nível superior com salário < salário médio da equipe a que ele pertence, ou projetos onde ao menos um membro da equipe é gerente do departamento que controla o projeto".

Esta consulta necessita da seleção de dois caminhos, o primeiro compreende projetos, equipe, nível e salário, e o outro compreende projeto, equipe e departamentos. Portanto, temos as seguintes seleções:

a) Caminho de Projeto a Empregado, via Equipe: M1 = σ_c(PESSOAL+PROJETO, <projeto, equipe, empregado>, {nome-proj, tem-salário:salário, nível, salário-médio}).

b) Caminho circular de Projeto a Projeto, via Equipe, Empregado e Departamento. M2 = σ_c(M, <*projeto*, *equipe*, *empregado*, *depart.*, controla, *projeto*, {nome-proj}). Todas as classes duplicadas de M1 e M2 são distintas por um índice. Por exemplo, *projeto*(1) em M1 e *projeto*(2) em M2. O esquema resultante do processo de seleção é M3 = M1 $\cup$ M2. Agora, especializamos M3, de forma a introduzir as condições da consulta:

M4 = ε(M3, *empregado*(1), *nível*='superior' $\wedge$ *salário*(*empregado*) < *salário-médio*(*equipe*))

De forma a executar a condição *ou* dentro da consulta, aplicamos uma *generalização* em *projeto*(1) e *projeto*(2): M5 = γ(M4, *projeto*(1), *projeto*(2))

Finalmente, o resultado é obtido por uma generalização de dois nomes de *empregado* e agrega estes nomes com *nome-proj*: ER = α(M5, γ(*nome*(1), *nome*(2)), nome-proj).

2) Consulta Temporal com *as-of*:

"Qual foi o primeiro salário dos empregados admitidos em 10.01.94?"

As-of (10.01.94): M1 = ϕn (PESSOAL, *empregado, begin(10.01.94*))

Resultado: ER = α(M1, *nome*, *salário*)

3) Consulta Temporal com *walk-through*:

"Qual o histórico salarial de todos os empregados?"

Walk-through(histórico): M1 = ωl (**PESSOAL**, *tem-salário*, *ever*)
Projeção Temporal: M2 = π((M1, *empregado* |*salário* |*ever*, *ever*))
Resultado: ER = α(M2, *nome*, *salário*)

4) Consulta combinando *walk-through* com *as-of*:

"Quais eram os salários dos empregados quando eles subiram de nível pela primeira vez?"

Encontre a primeira mudança de nível (*walk-through*): M1 = ωl (**PESSOAL**, *tem-nível, ever*)
Projeção Temporal: M2 = π((M1, *empregado* | *nível* |*ever*, *Last-instant(First-interval(int)))*
Encontre o salário (*as-of*): ϕn (M2, *tem-salário*, *at(t)*)

Capítulo 5

Um Estudo Comparativo

5.1. Introdução

Muitas interfaces gráficas para BD's foram desenvolvidas, em [Vadaparty93], estimava-se, até então, em torno de cinquenta interfaces gráficas, desde o surgimento do QBE [Zloof77]. Isto se deve ao fato de que muitos modelos de dados possuem uma representação visual. O desenvolvimento de tais interfaces também foi estimulado com o advento da tecnologia *workstation* com *displays bit-mapped* e dispositivos de apontamento (ex: *mouse*). Uma outra razão é que a visualização gráfica reduz o conhecimento cognitivo de um usuário de BD [Mohan93]. Mesmo nos modelos relacionais, uma forma mais visual de consulta tal como o QBE, tem provado ser superior à consulta especificada em SQL, tanto em termos de facilidade de acesso quanto à visualização conceitual.

A maioria das linguagens visuais desenvolvidas para BD's são interfaces gráficas para esquemas de BD's. Em geral, estas interfaces se enquadram dentro de uma categoria de classificação definida em [Mohan93], que será descrita posteriormente neste capítulo.

O capítulo é dividido em duas partes: a primeira parte vai apresentar os diferentes níveis sobre os quais um BD pode ser acessado, desde a linguagem natural até as linguagens de programação. A segunda parte apresenta três interfaces gráficas. Elas serão comparadas à interface ConTOM, segundo uma taxonomia baseada no trabalho descrito em [Hull87], que considera três pontos centrais: funcionalidade, ambiente de implementação e a utilização de gráficos. Cada ponto central vai apresentar diversas características com o objetivo de mostrar os aspectos comuns e diferenciados entre as interfaces, através da presença ou ausência destas características. Foram acrescidas algumas características, obtidas a partir de uma análise de outras interfaces gráficas, julgadas relevantes como critérios de avaliação de uma interface gráfica para BD's.

5.2. Níveis de Linguagem para BD's

Existem diferentes níveis sobre os quais um BD pode ser acessado, como mostra a figura 5.1. Em um nível mais alto está a **linguagem natural**. No nível abaixo, existem as **linguagens de consulta+** . São chamadas assim já que a maioria destas linguagens não apenas fornece facilidades de consultas, como também facilidades para atualização e manipulação de dados no BD. Elas são especificamente projetadas para o usuário **casual** - o usuário que apenas ocasionalmente acessa o BD, sem a necessidade de um aprendizado prévio de uma linguagem textual completa. As linguagens de consulta+ frequentemente apresentam recursos gráficos para o usuário poder utilizá-las. Estão também incluídos neste nível as linguagens *English-like* que são adequadas ao usuário que gostaria de acessar o BD mais frequentemente e está preparado para aprender uma linguagem simples. Em um nível mais baixo, encontram-se as **Linguagens de Manipulação de Dados** (LMD) que geralmente são utilizadas por um especialista em BD's. Por último, existem as **linguagens de programação** (C, Pascal, Cobol, etc) que são utilizadas por aplicativos de BD's, e interagem com uma LMD para acessar o SGBD.

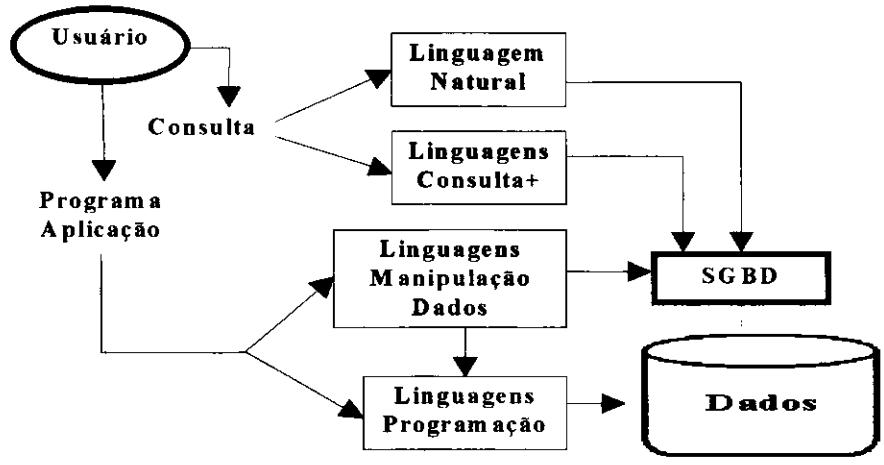


Figura 5.1. Linguagens de BD e o Usuário Final

Em adição à classificação por níveis, dois outros aspectos de linguagens são de interesse: o aspecto procedural e o não procedural. Uma linguagem é **não procedural** se ela permite ao usuário especificar o que ele deseja e não como obter o que ele deseja. Uma linguagem é **procedural** se o usuário tem que especificar os passos a serem seguidos para obter a informação que ele deseja. Portanto, em uma linguagem não procedural, uma consulta será especificada geralmente em um único comando, enquanto que em uma linguagem procedural geralmente será necessário especificar um conjunto de comandos para obter a resposta da consulta. Nas seções seguintes, exemplos de linguagens nos níveis da linguagem natural, gráfica e textual serão discutidas.

5.2.1. Interfaces em Linguagem Natural

Linguagem natural é por natureza informal, verbal e pode ser ambígua. Quando se usa linguagem natural, o usuário deve entender o contexto sobre o qual a linguagem natural está sendo utilizada, e pode deduzir o que é relevante mesmo quando algum detalhe ou fato foram omitidos. Por possuir estas características é que a comunicação através da linguagem natural se torna difícil. Outra desvantagem é, por exemplo, se a consulta formulada não é específica o suficiente ou é ambígua, um longo diálogo será necessário antes que o sistema "entenda" a consulta do usuário.

Codd considera as seguintes características importantes de uma bem-sucedida interface para BD's em linguagem natural:

- Um modelo de dados simples para o sistema.
- Uma linguagem *target* interna incorporando uma lógica de alto nível.
- Diálogo de esclarecimento dentro de um escopo limitado.
- Interrogação de múltipla escolha.
- Facilidade para o sistema criar novas definições baseado na consulta do usuário.
- Redeclaração da consulta do usuário em uma linguagem pseudo-natural.
- Formulação de consultas separado do acesso ao BD.

Vamos discutir com mais detalhes estas características. Uma vantagem da interface em linguagem natural é que os usuários não precisam saber nada sobre a estrutura lógica ou física do BD. Seguindo este raciocínio, Codd sugere que o modelo de dados sob a interface deveria ser simples, com poucos elementos estruturais, e sugere que o modelo relacional é adequado para este propósito.

Uma interface em linguagem natural, por um lado, tem que se comunicar com o usuário em uma linguagem informal. Por outro lado, deve comunicar-se com um sistema de computador em uma linguagem formal. Para facilitar esta tarefa, a linguagem *target* interna deve ser de alto nível, de forma que seja relativamente fácil traduzir de informal para formal e vice-versa. Isto implica que o *target* deve ser uma linguagem não procedural.

Com uma interface em linguagem natural, o usuário especifica suas exigências dentro da sua própria linguagem. Neste caso, um dicionário de dados é essencial, o qual não contém somente metadados padrão, como também informações adicionais tais como definições alternativas da linguagem, frases, etc, os quais podem ser associados aos elementos do BD. Isto ajudará a interface em associar componentes dentro da consulta com elementos do BD. Se a interface detecta um componente que ela não entende, ou está incompleto ou ambíguo, esta pode começar um diálogo com o usuário. Este diálogo, porém, será de escopo limitado, isto é, a questão que deve ser feita ao usuário será especificamente relacionada ao que a interface

entende da consulta e conhece do BD. Interrogações de múltipla escolha podem ser usados como uma alternativa para esclarecer o diálogo.

Quando uma palavra ou frase do usuário exigiu o uso de um diálogo esclarecedor e/ou interrogação, é importante que a palavra ou frase do usuário seja retida para uso futuro - assim não seria necessário o mesmo diálogo em uma outra ocasião. Portanto, a interface deve ser capaz de adicionar novas definições ao dicionário de dados, ou seja, ela deve aprender com a experiência.

Assim que a interface fosse capaz de gerar uma versão interna da consulta ela deveria convertê-la em uma linguagem pseudo-natural, apresentar esta consulta convertida ao usuário, e perguntá-lo se é o que ele realmente deseja. Neste instante, a linguagem *target* interna de alto nível é importante, desde que é muito mais fácil traduzir de uma linguagem não procedural de alto nível para uma linguagem pseudo-natural do que traduzir para uma linguagem de baixo nível. Se o usuário não gostou da redeclaração da sua consulta, novos diálogos serão necessários, seguido por outra redeclaração.

Durante a formulação da consulta, Codd sugere que apenas os metadados (informação do dicionário de dados) deveriam ser acessados, e só quando o usuário indica sua satisfação com a redeclaração da consulta, é que os dados devem ser acessados. Se o acesso aos dados fosse feito em estágios intermediários durante a formulação da consulta, pode-se consumir tempo e apresentar ao usuário muitos dados que ele não deseja. Contudo, às vezes o usuário não sabe o que quer, e prefere verificar os resultados e gradativamente refinar a consulta. Embora isto seja desejável, só funcionará bem com um BD relativamente pequeno. Portanto, a formulação da consulta será, geralmente, um processo completamente independente do acesso aos dados.

Um exemplo de um possível diálogo entre um usuário e uma interface em linguagem natural é ilustrado abaixo, extraído de [Codd78] (assuma que U = usuário, S = sistema e a data de hoje é 29/12/94):

U *Eu quero informação sobre livros a serem devolvidos hoje?*

S Com relação a "livros a serem devolvidos hoje", significa: "livros emprestados com retorno em 29/12/94? (Sim ou Não)

U *Sim*

S O que você quer sobre livros emprestados com retorno em 29/12/94?

1 Livro#

2 Catálogo

3 Nenhuma das respostas acima

Selecione um ou mais itens.

U *2*

S O que você quer sobre o catálogo?

1 Isbn

2 Autor

3 Título

4 Editora

5 Nenhuma das respostas acima

Selecione um ou mais itens.

U *2 3*

S O sistema entendeu a sua consulta como:

"Recupere o autor e o título do catálogo para todos os livros emprestados com retorno em 29/12/94"

1 Correto e completo

2 Ainda não completo

3 Incorreto

Selecione exatamente um item.

U *1*

5.2.2. Interfaces Gráficas

Uma das desvantagens da linguagem natural é a quantidade de diálogo necessário para construir a consulta, já que ela é verbal e ambígua. Uma alternativa para o usuário é uma interface gráfica. O usuário é capaz de "quebrar" uma consulta complexa em pequenas partes. Uma capacidade *full-screen* é essencial para as interfaces gráficas. Nas seções seguintes, as características principais de duas interfaces gráficas - QBE (*Query-By-Example*) e Cupid (*Casual User Pictorial Interface Device*), serão discutidas.

5.2.2.1. QBE

QBE (***Query-By-Example***) [Zloof77] foi projetada como uma interface de alto nível para um SGBD relacional. Ela é baseada na idéia de tabelas representando relações as quais são preenchidos com um "exemplo" para representar a consulta. Uma implementação importante do QBE está no sistema DB2 da IBM, onde QBE é fornecido como uma alternativa à linguagem SQL.

As características importantes de uma consulta QBE são: a tabela, o *elemento exemplo*, o *elemento constante* e o *elemento link*. No começo de uma consulta, o usuário seleciona as relações de interesse. O sistema fornece um menu de relações disponíveis e o usuário faz suas seleções sobre o menu. Depois da

seleção, QBE apresenta uma ou mais tabelas, cada tabela representando uma das relações selecionadas. A figura 5.2 apresenta uma tabela de Catálogos.

CATÁLOGOS	Isbn	Autor	Título	Editora	Ano
	...	.....	.....	.....	.....

Figura 5.2. Tabela QBE

O usuário, então, preenche em uma das colunas das tabelas com elementos de consulta QBE que são relevantes para a sua consulta. Por exemplo, suponha que ele deseja recuperar todos os títulos de todos os livros de Fernando Pessoa, o usuário preencheria a tabela como ilustrado na figura 5.3.

CATÁLOGOS	Isbn	Autor	Título	Editora	Ano
	...	Fernando Pessoa	p.xxx	.....	.....

Figura 5.3. Uma Simples Consulta QBE

A entrada de "Fernando Pessoa" na coluna Autor é um *elemento constante*; apenas as tuplas da relação que satisfazem o *elemento constante* será selecionado no resultado da consulta. Por *default*, a comparação é uma comparação "=", outras comparações devem ser explicitamente representadas. Mais de um *elemento constante* pode existir em uma consulta QBE, e estes por default são conectados por "*and*". O p. na coluna Título indica que os títulos dos livros selecionados devem ser impressos. O conjunto de atributos os quais podem ser marcados com p. representam o *target* da consulta. O xxx na coluna Título é indicado como um *elemento exemplo* e está sublinhado, indicando que o valor sublinhado não existe no BD.

Vamos retornar à consulta utilizada como exemplo na interface de linguagem natural descrita na seção anterior: "*Imprima o título e o autor de todos os livros a serem devolvidos em 29/12/94*". Esta consulta envolve múltiplas relações. A relação CATÁLOGOS é necessária para obter os títulos e autores, a relação EMPRÉSTIMOS é necessário para selecionar os empréstimos que serão devolvidos na data especificada, e desde que CATÁLOGOS e EMPRÉSTIMOS não estão diretamente relacionados, a relação LIVROS é também necessária para ligar as duas relações. A consulta é ilustrada na figura 5.4.

A consulta exige o uso do *elemento link*; um *elemento link* é um *elemento exemplo* que tem o mesmo valor nas relações a serem ligadas. No exemplo, o *elemento exemplo* na coluna Isbn em CATÁLOGOS é idêntico ao *elemento exemplo* na coluna Isbn em LIVROS, de uma forma similar, o *elemento exemplo* na coluna Livro# em LIVROS e EMPRÉSTIMOS. Estes pares de *elementos exemplo* são os *elementos link*, representando a condição de ligação.

CATÁLOGOS	Isbn	Autor	Título	Editora	Ano
	isbn	p.	p.	.....	.....

LIVROS	Livro#	Isbn
	b#	isbn

EMPRÉSTIMOS	Livro#	Pessoa#	Data-dev.
	b#		25/12/94

Figura 5.4. Uma Consulta Multi-Relação QBE

QBE é um exemplo de cálculo relacional orientado a domínio. Os *elementos exemplo* podem ser considerados variáveis que façam parte de domínios de atributos relevantes. Por exemplo, na figura 5.4, *o elemento exemplo* isbn faz parte do domínio Isbn dentro de CATÁLOGOS e LIVROS.

QBE suporta muitas outras facilidades, incluindo operadores agregados (*sum, count*, etc), agrupamento e um operador *all*. Ele também suporta definição de dados e atualização de BD's. Para definição de dados e atualizações, o usuário utiliza o mesmo mecanismo tabular. Maiores detalhes sobre QBE podem ser vistos em [Zloof77].

5.2.2.2. Cupid

Uma outra interface gráfica alternativa é o **Cupid** (*Casual User Pictorial Interface Device*) [McDonald75], baseado na álgebra relacional. Como o QBE, foi projetada como uma interface de alto nível para um SGBD relacional. Cupid, porém, não é de formato tabular, ele utiliza símbolos para representar componentes de uma consulta. O usuário seleciona componentes relevantes e constrói sua consulta a partir destes componentes.

Os principais símbolos utilizados pelo Cupid são mostrados na figura 5.5.

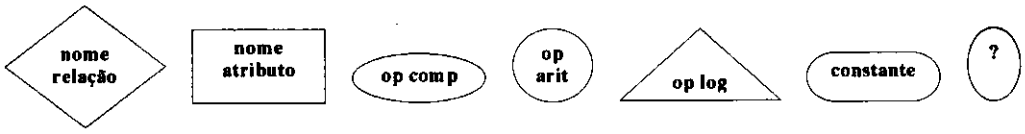


Figura 5.5. Principais Símbolos do Cupid

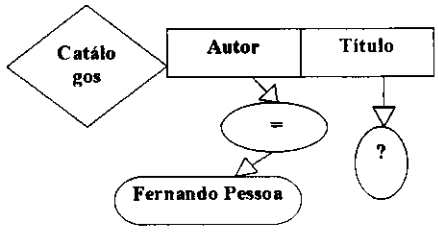


Figura 5.6. Uma Simples Consulta Cupid

Em adição aos símbolos acima, um conector é usado para indicar ligações entre relações e um símbolo de operador agregado é também fornecido. A figura 5.6 mostra uma simples consulta envolvendo uma relação. A consulta é novamente

encontrar os títulos de todos os livros de Fernando Pessoa.

A consulta ilustra algumas características de Cupid. Primeiro, o usuário especifica somente aqueles atributos relevantes para a consulta, no exemplo, *Autor* e *Título*. Segundo, o círculo com ? representa a lista *target*, e o operador "=" e a constante "Fernando Pessoa", ligados ao atributo *Autor*, representam a condição de seleção.

A consulta multi-relação para imprimir título e autor de todos os livros emprestados com retorno em 29/12/94 é ilustrado na figura 5.7. A figura mostra como os conectores e operador "=" são usados para representar a condição de ligação, e pode ser comparado com a consulta QBE correspondente à figura 5.4.

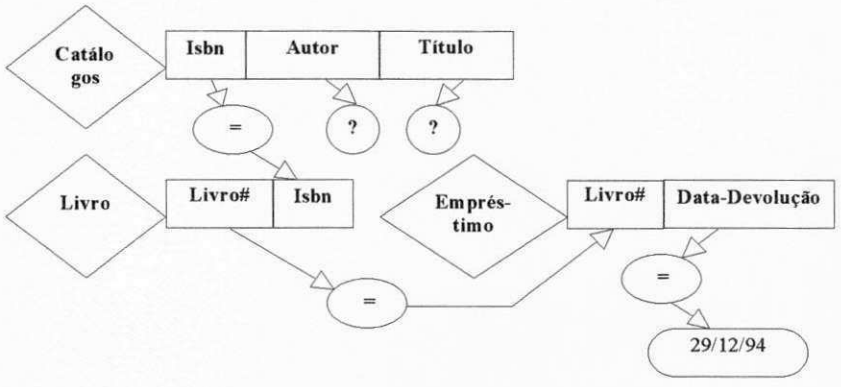


Figura 5.7. Uma Consulta Multi-relação Cupid

5.2.2.3. Vantagens e Desvantagens das Interfaces Gráficas

As interfaces gráficas têm inúmeras vantagens para o usuário casual. Primeiro, elas minimizam a quantidade de esforço de *typing* necessário; segundo, elas possibilitam ao usuário construir sua consulta especificando seus componentes em qualquer ordem; e terceiro, o usuário tem uma visão da sua consulta na tela como ela está sendo construída, e pode adicionar/retirar componentes da consulta até que a visão corrente represente a consulta que ele deseja. Outra vantagem adicional das interfaces gráficas é a possibilidade de fazer o sistema amigável ao usuário através de dispositivos de apontamento sofisticados (*mouse*, etc). Contudo, interfaces gráficas tem desvantagens. Em interfaces para o sistema relacional, o usuário precisa entender sobre relações e como ligá-los, além de saber alguma coisa sobre a estrutura do BD (isto não é necessário em uma interface em linguagem natural). Um terceiro problema é a necessidade de uma tela gráfica grande, telas de tamanho padrão são realmente muito pequenas para interfaces gráficas sofisticadas.

5.2.3. Linguagens de Consulta Textuais

Linguagens de consulta textuais são adequadas a diferentes classes de usuários. Linguagem natural e interfaces gráficas são mais adequadas a usuários casuais. Os usuários regulares e mais experientes, porém, podem achar tais interfaces tediosas de usar. Eles sabem exatamente que informação eles querem obter do BD e como expressar as consultas relevantes. A principal vantagem de uma linguagem de consulta textual formal para tais usuários é que não existe diálogo nem qualquer necessidade de mover cursores sobre a tela.

Nesta seção, será apresentada a linguagem de consulta SQL, projetada para sistemas relacionais e baseada no cálculo relacional.

5.2.3.1. SQL

SQL (*Structured Query Language*) [Chamberlin74] é a linguagem adotada pelo projeto System R da IBM, que foi um protótipo de pesquisa, base dos sistemas SQL/DS e o DB2. SQL é uma linguagem padrão para gerenciadores relacionais e é oferecida pela maioria dos SGBD's atualmente no mercado.

A sintaxe básica de uma consulta SQL é a seguinte:

```
SELECT < lista target>
        FROM <lista de relações>
        WHERE <condição>
```

Para recuperação de atributos selecionados de uma relação, a cláusula WHERE é omitida; por exemplo:

```
SELECT Título
        FROM CATÁLOGOS
```

No caso de recuperação envolvendo múltiplas relações, a condição do WHERE será composta de um componente de ligação juntamente com um componente de seleção, se relevante. Porém, o componente de ligação pode ser um consulta SQL aninhada, por exemplo:

```
SELECT ... FROM ...
        WHERE ... =
                (SELECT ... FROM ...
                        WHERE ...)
```

As consultas seguintes correspondem aos exemplos anteriores:

Exemplo1. "Encontre os títulos de todos os livros de Fernando Pessoa"

```
SELECT Título
    FROM CATÁLOGOS
    WHERE Autor = "Fernando Pessoa"
```

Exemplo2. "Encontre o título e autor de todos os livros emprestados com retorno em 29/12/94" (nomes de atributos são prefixados pelos nomes de relações relevantes se eles não são únicos na consulta):

```
SELECT Título, Autor
    FROM CATÁLOGOS, LIVROS, EMPRÉSTIMOS
    WHERE CATÁLOGOS.Isbn = LIVROS.Isbn
        AND LIVROS.Livro# = EMPRÉSTIMOS.Livro#
        AND Data-devolução = "29/12/94"
```

No *exemplo2*, note o uso de duas cláusulas na condição de ligação, uma ligando CATÁLOGOS com LIVROS e outra ligando LIVROS com EMPRÉSTIMOS.

A sintaxe SQL do *exemplo2* não é a única forma de especificar a consulta. Esta consulta particular fornece um exemplo para ilustrar o uso de consultas aninhadas em SQL. A lista *target* contém somente atributos da relação CATÁLOGOS e, portanto, a consulta básica pode ser considerada como envolvendo apenas esta relação. A versão aninhada da consulta é a seguinte:

```
SELECT Título, Autor
    FROM CATÁLOGOS
    WHERE Isbn IN
        (SELECT Isbn
            FROM LIVROS
            WHERE Livro# IN
                (SELECT Livro#
                    FROM EMPRÉSTIMOS
                    WHERE Data-devolução = "29/12/94"))
```

Uma consulta aninhada SQL incorpora características de álgebra relacional, com operações de junção implícitas no ponto de aninhamento. Uma subconsulta interna é processada primeiro e retorna um conjunto de constantes os quais são utilizadas na subconsulta externa.

Entre outras facilidades oferecidas pelo SQL estão as facilidades de definição e manipulação de dados, funções *group by, order by*, declarações de atualização e deleção usando uma sintaxe similar à declaração SELECT, e declaração de inserção. SQL também suporta visões de usuário. Adicionalmente, nas implementações de SQL mencionadas previamente, DB2 e ORACLE, SQL pode ser embutido em uma linguagem hospedeira para programas de aplicação.

A flexibilidade do SQL tem vantagens e desvantagens. Uma das vantagens é que o usuário pode escolher uma técnica que seja mais adequada para ele, porém, possuir numerosas formas de especificar uma consulta pode também confundir o usuário. Outra principal desvantagem é que o usuário tem dificuldades que envolve memorização de sintaxe, memorização de esquemas e complexidade em reconhecer relacionamentos do esquema de BD antes de especificar as consultas.

Para finalizar, entre outras linguagens de consulta textuais, destacamos o QUEL[Stonebraker76], uma outra linguagem similar ao SQL, também projetada para sistemas relacionais e com as mesmas vantagens e desvantagens; a linguagem QRS, linguagem projetada para sistema de redes e Prolog[Bratko86], uma linguagem lógica baseada no cálculo de predicados, onde fatos e regras estão combinadas juntas em um programa.

5.3. Aspectos Comparativos de Algumas Interfaces Gráficas

Nesta seção, vamos apresentar uma breve descrição de três interfaces gráficas: SNAP[Bryce86], VQL[Mohan93], Graqula[Sockut93]. Logo após, um critério comparativo destas interfaces com ConTOM será discutido.

5.3.1. O Sistema Snap

O sistema **SNAP** (***Schemas Notated As Pictures ou Semantic Navigation And Perusal***) [Bryce86] é um gerenciador de esquemas para o modelo de dados IFO [Abiteboul84] e fornece um suporte para projeto de esquemas, *browsing* de esquemas e especificação de consultas. O modelo de dados IFO é um modelo semântico orientado a objetos, que fornece características para a representação de objetos simples e complexos, relacionamentos funcionais e relacionamento É-UM. Na definição formal de IFO [Abiteboul84], esquemas de BD's são grafos dirigidos com certas propriedades; este grafos fornecem uma estrutura dentro da qual diversos tipos de relacionamentos de dados são visualmente representados.

Existem dois princípios fundamentais dentro do IFO, o primeiro princípio é o uso de "representações fragmentadas" como blocos de construção básica para esquemas. Cada representação fragmentada (ou simplesmente "rep") descreve um conjunto de entidades, onde a cada entidade estão associados todos os

seus atributos. O segundo princípio de IFO é o uso de representações distintas (nodos distintos dentro do esquema gráfico) para papéis distintos de uma mesma entidade. Estes dois princípios permitem ao sistema SNAP fornecer tanto uma visão modular quanto níveis de abstração na representação dos esquemas.

A figura 5.8 ilustra os dois princípios através de uma parte de um esquema de uma agência de turismo (o esquema completo pode ser visto em [Bryce86]). Esta figura ilustra três representações fragmentadas: CIDADE, representando as cidades sobre as quais as agências de turismo organizam "*tours*"; PACOTE, representando os "pacotes turísticos"; e HOTEL, representando os hotéis utilizados pela agência. Note que a entidade *cidade*, representada por um losango, indica que cidade é um tipo abstrato. Os conectores que partem deste losango, ligando a outros nodos, correspondem aos atributos de cidade. Os retângulos indicam que as entidades são de tipos imprimíveis. O nodo *clima* representa um nodo agregado e o nodo *cnome* serve como um identificador para cidade, representado pelo símbolo ⇔.

Considerando agora o "rep" PACOTE, a entidade *pacote* possui um atributo, o atributo *cidade*, indicando as cidades do pacote turístico. O **atributo** *cidade* está representado como um círculo, ligando este atributo à **entidade** *cidade*, através de uma ligação *é-um*, representada pelo símbolo ⇒ (no modelo IFO, existem duas formas de representar a ligação *é-um*, a primeira forma equivale à ligação É-UM dos modelos semânticos entre tipo e subtipo, e a segunda forma está sendo exemplificada entre a entidade *cidade* e o atributo *cidade*). O nodo circular, como parte do "rep" PACOTE, ilustra o segundo princípio.

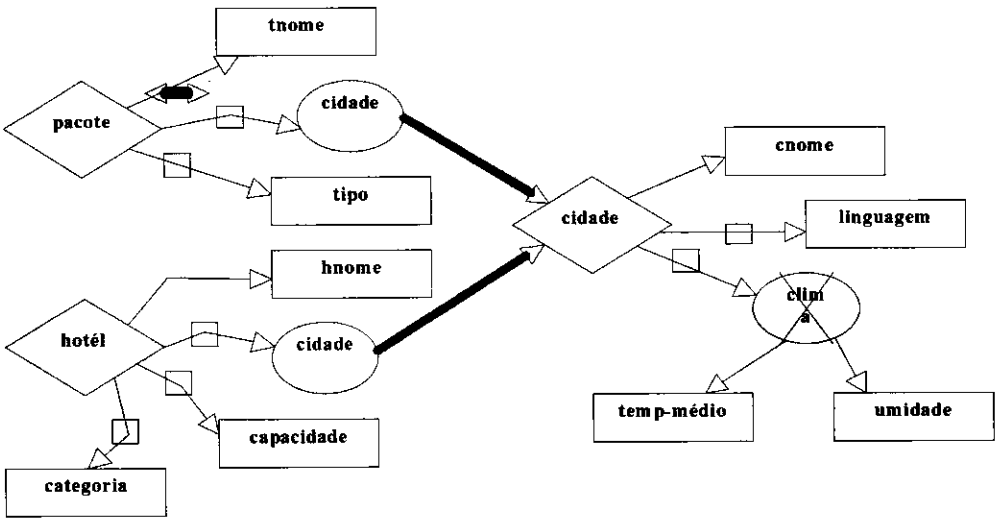


Figura 5.8. Três Representações Fragmentadas

A representação de nodo circular é necessária para grandes esquemas, pois, sobre os quais, seria confuso muitas ligações para um único nodo. O sistema SNAP permite ao usuário ocultar o relacionamento *é-um*, não afetando a visualização dos "reps". Portanto, um "rep" em SNAP serve como uma representação visual de "pedaços" de tamanho razoável do esquema total e é correspondente a um módulo inicial em ConTOM (nodos representando as cidades equivalem ao nodo "duplicado" definido no capítulo 3, seção 3.3.3.2).

Além da representação modular de esquemas, similar ao ConTOM, o sistema SNAP contribui na área de especificação gráfica de consultas. SNAP fornece uma sintaxe bidimensional para consultas, através da combinação de aspectos da filosofia de QBE e Cupid juntamente com a abordagem gráfica para representação de esquemas, encontrada em modelos semânticos.

Na especificação de uma consulta, o usuário interage com três tipos de janela: janela de esquema, janela de consulta e a janela de resposta. Mais de uma janela de consulta pode estar associada a um esquema e mais de uma janela de resposta pode estar associada a uma janela de consulta. Consultas SNAP são expressas usando grafos de consulta que são formados pela combinação de um ou mais "reps". Um grafo de consulta pode conter vários "reps" de consulta. Os valores associados com os "reps" de consulta são especificados através de três mecanismos fundamentais: **restrição de nodo**, permitindo ao usuário associar uma restrição diretamente a um dado nodo de um "rep"; **arco comparador**, que permite ao usuário indicar que um certo relacionamento deve permanecer entre dois valores associados a diferentes nodos, e o terceiro mecanismo permite ao usuário construir **novas funções** a partir das existentes de uma maneira totalmente visual e incluí-los em "reps" de consulta.

A figura 5.9 ilustra três janelas de consulta representando os seguintes exemplos (cada exemplo apresenta o mecanismo utilizado em SNAP): *"Liste todos os hotéis de S. Paulo os quais possuem uma capacidade > 250", "Liste o par turista-cidade tal que existe um hotel nesta cidade cuja categoria seja maior ou igual ao conforto do turista"* e *"Liste todos as viagens cujo número de linguagens necessárias para as mesmas devem ser menor ou igual ao número de linguagens que o guia deve conhecer"*. A figura dentro do quadrado é uma parte do diagrama de esquema, composta somente por um caminho de nodos feita pelo usuário, com o objetivo de gerar a janela de consulta ao lado, referente ao 3o exemplo. Os nodos em destaque indicam que eles devem aparecer na resposta. Apesar de alguns nodos em destaque não serem imprimíveis, SNAP automaticamente apresenta no resultado os valores dos atributos identificadores destas entidades abstratas (não imprimíveis).

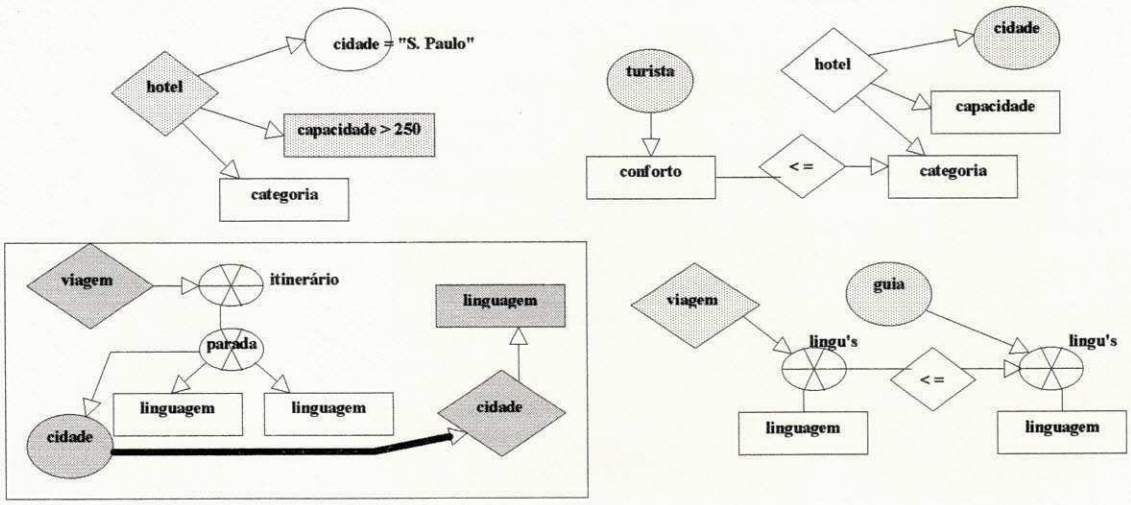


Figura 5.9. Janelas de Consulta dos Três Exemplos

Para finalizar, quando uma consulta é especificada, o sistema SNAP escolhe um formato *default* para a resposta (relação na 1a. forma normal), ele também fornece um mecanismo de diálogo sobre o qual o usuário poderá escolher outros formatos.

5.3.2. A linguagem VQL

A linguagem VQL (*Visual Query Language*) [Mohan93] é uma linguagem que interage com o modelo de dados orientado a objetos SSONET (*Structured Semantic Object Network*) [Mohan90], um modelo que foi projetado para modelagem de domínios esquema-intensivos, ou seja, domínios que contém um grande número de classes e muitos interrelacionamentos entre as classes. Uma grande proporção das consultas é especificada a nível de esquema, contudo, consultas a nível de classe e a nível de instância também podem ser especificadas.

Cada consulta VQL é graficamente especificada utilizando um conjunto de primitivas visuais. Existe uma sintaxe formal e uma estrutura semântica sob a criação de uma consulta gráfica em VQL. As primitivas gráficas são mapeadas diretamente sobre primitivas de consulta textuais que são expressões de predicados (similar aos predicados lógicos usados em linguagens tais como Prolog). Sempre que primitivas visuais múltiplas são combinadas graficamente para formar uma consulta visual, ocorre uma representação paralela, como expressão lógica do predicado. Uma gramática BNF [Naur63] formal foi desenvolvida para a criação de expressões de consulta válidas, dentro da linguagem de predicados.

O modelo de dados SSONET [Mohan90] é composto de três níveis conceituais: o nível abstrato, que captura o conhecimento genérico do domínio; o nível que corresponde diretamente ao nível esquema em BDOO's e o nível mais baixo, o nível de instância, que corresponde ao nível de dados em BD's. Existem cinco tipos fundamentais de objetos definidos dentro da estrutura do modelo: *dbClass, dbLinkType, dbInstance, dbClassLink* e *dbInstanceLink*. Os dois níveis mais altos consistem de *dbClasses, dbClassLinks* e *dbLinkTypes*, este último é restrito somente ao nível abstrato. *DbInstance* e *dbInstanceLink* correspodem ao nível de instância.

Um *dbClass* é uma representação explícita de propriedades estruturais e comportamentais de qualquer objeto. Um *dbClassLink* é uma representação explícita de qualquer tipo de relacionamento que pode existir entre *dbClasses*. Cada *dbClassLink* é uma instanciação específica de um *dbLinkType* particular, ou seja, *dbLinkTypes* são definições de tipos abstratos de relacionamentos que podem existir entre *dbClasses*. Por simplicidade, vamos apresentar somente três tipos de *dbLinkTypes*: o *dbClassLink* "I" (Interação), que é um *dbLinkType* mais genérico e não direcional e corresponde a um relacionamento comum. O *dbClassLink* "G" (Generalização) que corresponde a uma hierarquia de generalização, e o *dbClassLink* "A" (Agregação) que corresponde a uma hierarquia de agregação. Assim como *dbClassLink* é uma instanciação específica de um *dbLinkType*, um *dbInstance* é uma instanciação específica de um *dbClass* particular. Além da

necessidade de representar *dbInstances*, existe a necessidade de capturar informação sobre qual *dbInstance* tem relacionamento com outros *dbInstances*. Isto é alcançado através dos *dbInstanceLinks*, que são ligações entre *dbInstances* específicos.

A figura 5.10 ilustra um esquema dentro do modelo SSONET, utilizando o exemplo do sistema de agência de empregos (ver capítulo 3, seção 3.3.2).

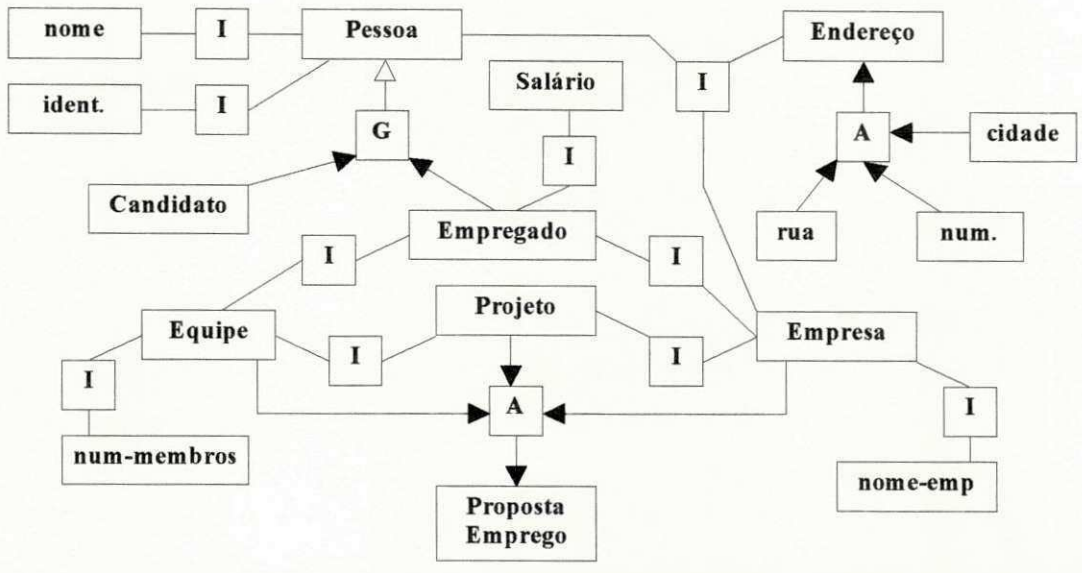


Figura 5.10. Exemplo de um Esquema em SSONET

A linguagem VQL é construída dentro de três primitivas gráficas básicas mostradas na figura 5.11. Elas podem ser usadas na criação de vários tipos de consultas sobre um BD ou base de conhecimento. Em geral, as primitivas visuais icônicas representam objetos lógicos de BD's.

A primitiva de consulta da figura 5.11(a) é chamada de **Ícone Classe**, utilizada para referenciar qualquer classe de BD particular (*dbClass*), a parte superior desta primitiva serve para especificar o nome da *dbClass*, enquanto que a outra parte serve para referenciar qualquer instância particular da classe.

A primitiva de consulta da figura 5.11(b) é o **Ícone Atributo** que corresponde às variáveis de instância de um *dbClass* particular e é composto de duas partes: a parte superior especifica o nome do atributo e a parte inferior pode ser um valor de atributo, um domínio de valores do atributo ou uma restrição sobre o valor do atributo.

A primitiva da figura 5.11(c) é o **Ícone Ligação**, que referencia um conjunto especial de classes que correspondem aos diversos objetos de ligação (*dbClassLinks* e *dbInstanceLinks)*. A parte central corresponde a um nome de um *dbclasslink*, a parte inferior referencia esta classe em particular e a parte superior serve para indicar o tipo de ligação (*dbLinkType*).

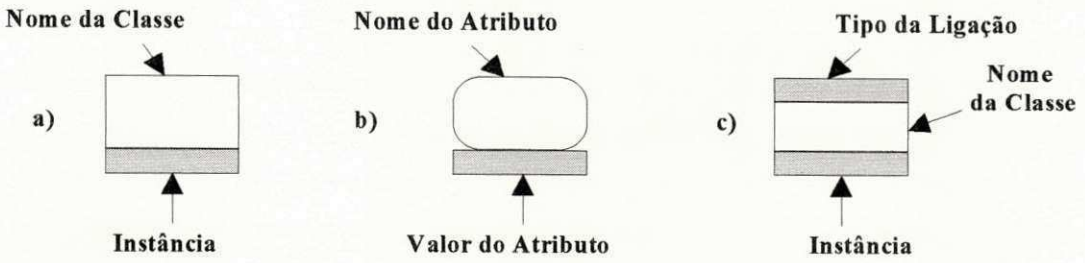


Figura 5.11. Primitivas Visuais de Consulta

Vamos agora especificar exemplos de consultas em VQL. Estas consultas ilustram a sintaxe básica e a semântica associada com a linguagem.

Considere o seguinte exemplo: *"Encontre todos os nomes dos empregados com salário > 1000 reais"*. O único objetivo desta consulta é acessar instâncias de uma simples *dbClass* que satisfaz uma certa restrição. Esta consulta pode ser graficamente representada usando primitivas visuais como mostrado na figura 5.12(a). Vamos considerar agora um exemplo de uma consulta com junção: *"Encontre os nomes das empresas que possuem empregados com salário > 5.000 reais"*. Consultas deste tipo devem ser especificadas usando a primitiva de Ligação para o *dbLinkType* "Interação", como ilustra a figura 5.12(b).

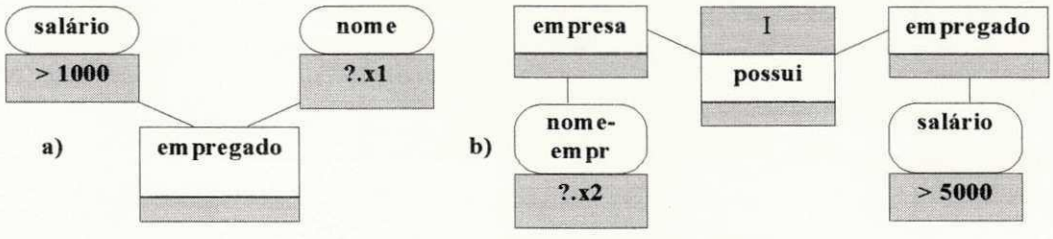


Figura 5.12. Exemplos de Consultas VQL

Para finalizar, VQL tem o poder de expressar consultas complexas, além do usuário poder especificar consultas recursivas, quantificação universal, negações, exigindo apenas um pequeno esforço do usuário em criar consultas visuais.

5.3.3. A Linguagem Graqula

A linguagem Graqula (***Graphical query language***) [Sockut93] é uma linguagem gráfica para consulta e atualização de BD's. Graqula fornece uma interface de usuário para o modelo E-R (Entity-Relationship) [Chen76]; outra versão fornece uma interface de usuário para o modelo relacional [Codd70] (vamos discutir aqui somente a versão para o modelo E-R). Além das facilidades de consulta que Graqula oferece para BD's, existem outras facilidades tais como: definição e *browsing* de esquemas, geração de relatórios, etc.

Basicamente, a funcionalidade de Graqula é a seguinte: a informação aparece como uma janela sobre a tela. O usuário começa identificando o BD durante a criação de uma **janela de esquema**, o qual contém um **diagrama de esquema**. O usuário visualiza as entidades e os relacionamentos dentro do diagrama de esquema e então pode criar **janelas de consulta**; cada janela de consulta contém um **diagrama de consulta** especificado pelo usuário. Em uma janela de consulta, o usuário adiciona detalhes tais como projeções e condições. A consulta pode ser executada dentro da janela de consulta e Graqula apresenta o resultado como uma tabela dentro de uma **janela de resultado,** ou envia para um arquivo especificado pelo usuário, ou para a impressão. A figura 5.13 mostra uma hierarquia de tipos genéricos de objetos gráficos que fazem parte do Graqula (vamos descrever brevemente alguns componentes desta hierarquia):

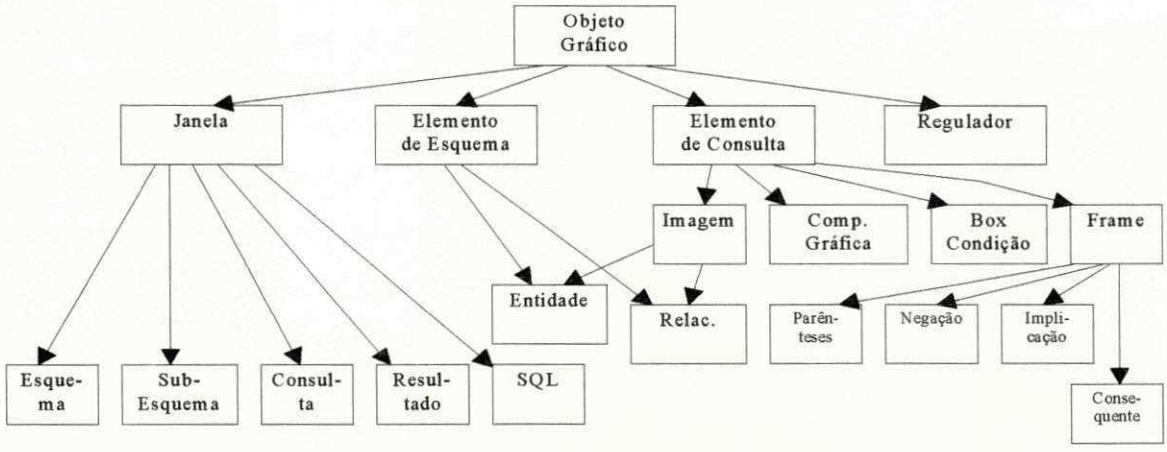


Figura 5.13. Hierarquia Genérica de Tipos de Objetos Gráficos

Uma **janela SQL** apresenta uma consulta em SQL. Um diagrama de esquema descreve os **elementos de esquema** (entidade e relacionamento). A figura 5.14 mostra um exemplo de esquema bem geral, pois os atributos que fazem parte das entidades não aparecem, contudo, o usuário pode expandir o esquema apresentando os seus atributos.

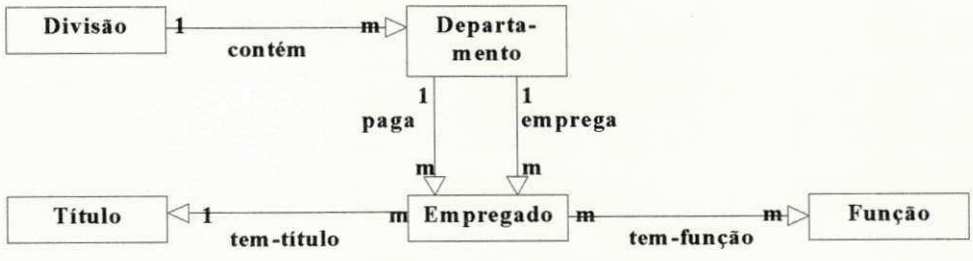
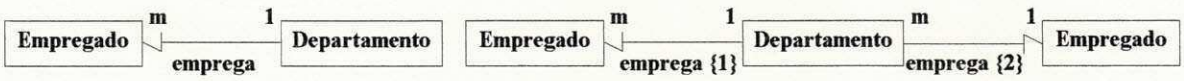


Figura 5.14. Um Diagrama de Esquema

Um diagrama de consulta descreve os **elementos de consulta**. Uma **imagem** (imagem entidade ou relacionamento) é parte de um diagrama de consulta e indica a participação de uma entidade ou relacionamento naquela consulta. Uma **comparação** (ex: junção) pode ser **gráfica** ou **textual**. Um ***box* de**

condição especifica uma combinação de operações lógicas (conjunção, disjunção e negação) sobre condições textuais. Um ***frame*** (parênteses, negação, implicação, ou *frame* consequente) especifica uma operação lógica e quantificação para elementos de consulta. A implicação tem dois operandos: o antecedente **implica** o consequente, desta forma, a implicação utiliza um par de frames: *I-frame* e *C-frame*. O par, na verdade, significa "(... → ...)" na linguagem textual e "*if* ..., *then* ..." em Inglês. Um **regulador** aparece em uma janela de consulta e controla aspectos da consulta relacionados a formato, execução ou salvamento da mesma.

A figura 5.15(a) é um diagrama de consulta (não uma consulta completa) para listar os empregados do departamento de Pessoal. Um diagrama contém muitas imagens de um elemento de esquema particular, se a consulta utiliza aquele elemento de muitas maneiras. Por exemplo, para listar os empregados do departamento de Ana Silva, usamos a entidade Empregado para listar empregados e para especificar Ana Silva, como ilustra a figura 5.15(b) (os sufixos {1} e {2} são gerados pelo sistema, se o elemento do esquema repete as imagens).



a) b)

Figura 5.15. Diagramas de Consulta

Após a especificação de um diagrama de consultas, o usuário pode expandir imagens (mostrar atributos), adicionar outros elementos de consulta, modificar o diagrama, adicionar projeções, condições e atualizações em imagens expandidas, adicionar expressões agregadas, operações lógicas e quantificação. Para cada imagem, o usuário pode mudar entre um formato compacto (inicial) e um expandido. Imagens de entidade expandidas incluem *rows*, um *heading* para *rows* e *scroll bars* horizontal e vertical. Um *row* contém um nome de atributo, espaço para o operador de comparação e espaço para o operando direito.

O usuário pode adicionar qualquer número de condições textuais sobre atributos dentro das imagens de entidade expandida e *boxes* de condição, através da entrada de um operador na coluna OP dentro da mesma linha daquele atributo, e através da entrada de um operando direito na coluna Valor (o OP sem operador indica que ele é "=", e não havendo preenchimento nas duas entradas, indica que não há condição). Por *default*, uma imagem de entidade expandida representa a **conjunção** de condições textuais dentro dela.

A projeção em Graqula se realiza através do "*click*" sobre um nome de atributo em uma imagem de entidade expandida. Os nomes dos atributos projetados aparecem em vídeo reverso (um "*click*" sobre o nome da imagem indica a projeção de todos os atributos).

A figura 5.16 ilustra um exemplo de consulta que lista o nome e salário de cada empregado cujo salário > 5000 e cujo ano de admissão corresponde ao ano de formação da divisão de Pesquisa (os *scroll bars* não

aparecem na figura e "(and)" dentro da coluna Valor indica a operação lógica *default* da imagem). A consulta utiliza condições textuais sobre atributos e uma junção que compara atributos de duas imagens de entidade. O operando direito da condição pode ser uma expressão, o qual envolve especificação de atributos, constantes, agregados, parênteses, operadores aritméticos e operadores de *string*. Graqula também suporta comparações gráficas entre atributos de diferentes imagens de entidade.

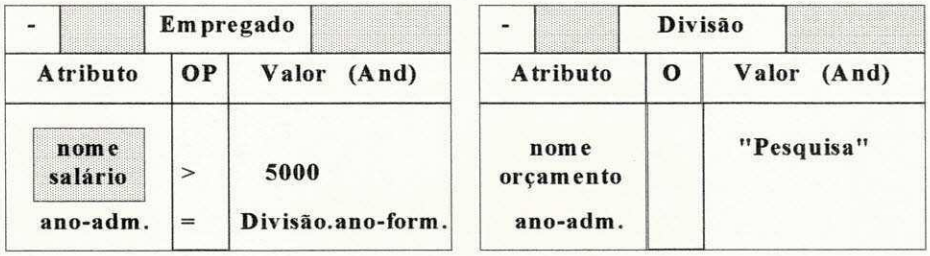


Figura 5.16. Exemplo de uma Consulta com Junção

Existem consultas que envolvem imagens de relacionamento. A figura 5.17 ilustra uma consulta que lista o nome de cada empregado que trabalha no departamento que paga aquele empregado (note que é um exemplo que usa relacionamento cíclico).

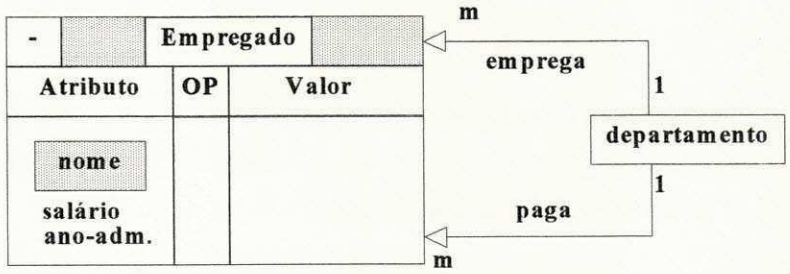


Figura 5.17. Consulta com Imagem de Relacionamento

Adicionalmente, na versão relacional do Graqula, ocorre a geração de consultas SQL para execução externa à Graqula, e o usuário pode verificar a consulta SQL gerada dentro da janela SQL. Outras facilidades de Graqula são: operações lógicas para condições textuais, quantificação, implicação, negação, operadores agregados, atualizações do BD, mapeamento de consultas E-R para relacionais, salvamento de consultas, ordenação e salvamento de resultado.

5.3.4. Aspectos Comparativos

A comparação entre o ambiente ConTOM e as interfaces gráficas descritas nas seções anteriores será baseada na taxonomia de [Hull87] considerando três pontos centrais: funcionalidade, ambiente de implementação e a utilização de gráficos, além de outros critérios comparativos, que serão introduzidos.

5.3.4.1. Funcionalidade

A Funcionalidade compreende: visualização e acesso ao esquema (*browsing*), definição de esquema, especificação de consulta e *browsing* de dados.

Considerando a **visualização** de um esquema, o sistema SNAP é semelhante ao ambiente ConTOM na representação das hierarquias de abstração (generalização, agregação e agrupamento) e na possibilidade do esquema ser dividido em módulos (os "reps" em SNAP são similares aos módulos iniciais em ConTOM). Contudo, os "reps" não derivam outros "reps" através das hierarquias de abstração; SNAP não apresenta as hierarquias de abstração em "reps" distintos, ou seja, não existe uma hierarquia entre módulos, como existe em ConTOM. Por apresentar esta característica, a visualização do esquema em ConTOM é mais completa, apresentando um esquema como uma estrutura modular e a interdependência entre os módulos, através da qual o usuário pode obter um *zoom* de cada módulo (ver capítulo 3, seção 3.3.4).

Comparando a interface VQL com ConTOM, a nível de visualização, VQL não representa visualmente a hierarquia de agrupamento e não fornece uma visualização modular do esquema tornando-se portanto, de mais difícil visualização para esquemas complexos. Contudo, existe um aspecto interessante em VQL que está ausente em ConTOM, na visualização de um esquema, a interface apresenta uma interpretação semântica dos componentes gráficos.

Em Graqula, baseado no modelo E-R, não há representação visual de nenhuma hierarquia. Existem dois níveis de visualização sobre o esquema. No nível mais alto, somente as entidades e os relacionamentos entre as mesmas são visualizados. Os atributos de cada entidade aparecem no nível mais baixo.

Considerando o mecanismo de **acesso** (*browsing*), em SNAP, pode-se acessar todos os nodos localizados em "reps" distintos e que estão relacionados pela ligação *é-um* (a ligação *é-um* neste contexto equivale a nodos que possuem diferentes papéis e referenciam a mesma entidade, ver seção 5.3.1). Em ConTOM, o mecanismo de acesso é mais flexível, pois além de localizar nodos que estão "distribuídos" em vários módulos, pode-se localizar nodos que estão relacionados pelas hierarquias de abstração através da seleção sobre os mesmos.

O acesso a um esquema em VQL é igual ao acesso em um ECG não modular em ConTOM. Além do acesso a nível de esquema, existe o acesso a nível de instância, que corresponde ao nível de dados de um BD.

Em Graqula, o acesso ao esquema pode ser visual ou através de diretórios. Os diretórios surgem como uma forma alternativa para grandes esquemas e que substitui a abordagem modular. O diretório lista tipos de entidade, tipos de relacionamento e atributos. Para cada atributo, há uma lista de tipos de entidade que o contém. Em grandes esquemas, eles aparecem com recursos de *scroll bar*.

Considerando a **definição** de um esquema, o ambiente ConTOM é o único que se baseia em uma metodologia (FADO ou POKER) como processo de geração de um esquema conceitual e é o mais completo na obtenção das informações de cada objeto gráfico, componente de um esquema. No sistema SNAP, a criação dos objetos gráficos é feita através de menus *pop-up* para indicar o tipo de objeto (abstrato, imprimível, etc) e *box pop-up* para texto, como identificação do objeto gráfico (o SNAP é similar ao Cupid desde que muitos dos nodos de esquemas IFO desempenham o mesmo papél como componentes de uma tupla no modelo relacional). O artigo sobre o VQL[Mohan93] não descreve como um esquema no modelo SSONET é definido pelo usuário. Em Graqula, o usuário deve estar familiarizado com o modelo E-R, na definição de um esquema.

Considerando a **especificação de uma consulta,** o sistema SNAP é similar ao ConTOM no processo de seleção. Em SNAP, combina-se um ou mais "reps" (módulos), seleciona-se os nodos de interesse, e gera-se uma janela de consulta como uma etapa prévia para aplicação dos seus operadores de consulta. Após estes procedimentos, a filosofia de SNAP, que combina aspectos de QBE e Cupid, é totalmente diferente da utilizada por ConTOM, que segue a filosofia de especificar uma consulta como se estivesse criando novas classes dentro das hierarquias de abstração. Em SNAP, não se deriva um novo módulo contendo novas classes a cada aplicação de um operador, como em ConTOM; o seu grafo de consulta apresenta uma conjunção explícita dos resultados dos mecanismos que ele apresenta, tornando-se, portanto, mais simples para o usuário **visualizar** a consulta como um todo. Por outro lado, o processo de **gerar** uma consulta em ConTOM se torna mais "intuitivo", pois a especificação de uma consulta equivale à construção de um ECG modular, ou seja, tanto a criação de um esquema, quanto à consulta estão dentro do mesmo ambiente interativo.

Considerando os mecanismos de consulta utilizados em SNAP, a projeção e agrupamento são bem mais simples do que em ConTOM, pois não há necessidade da intervenção de operadores; os nodos projetados aparecem em destaque e o agrupamento de dados em SNAP faz parte do resultado da consulta, e não na especificação, como em ConTOM. Um mecanismo em SNAP, similar ao ConTOM, é a criação de novas funções, onde o usuário ativa um "caminho" de nodos intermediários em um esquema, com objetivo de associar dois nodos não ligados diretamente. ConTOM não adotou este último como um mecanismo de operação e sim como um mecanismo funcional utilizado em todos os seus operadores de consulta.

Em VQL, o processo interativo durante a especificação de uma consulta é diferente de ConTOM. Os operadores da consulta em VQL são representados **explicitamente** como primitivas visuais icônicas (ver seção 5.3.2). O usuário faz uma combinação destas primitivas visuais, baseando-se nos componentes do esquema conceitual SSONET para a criação de uma consulta. Em ConTOM, os operadores de consulta são construídos **diretamente** dentro do esquema conceitual. Por exemplo, a restrição de um nodo em VQL é feita gerando a primitiva correspondente ao ícone atributo; dentro desta primitiva, inclui o nodo a ser

especializado e o predicado que indica a restrição. Em ConTOM, a restrição de um nodo é feita selecionando-o e aplicando o operador de especialização.

A especificação de consulta em Graqula possui uma etapa similar à etapa de seleção em ConTOM, onde ocorre a seleção dos componentes de interesse para a consulta, dentro do diagrama de esquema, para depois serem expressas as condições e projeções. A única diferença dentro deste processo inicial, é que em Graqula, em vez de seleção sobre os elementos de esquema de interesse, o usuário deve movimentá-los para a janela de consulta, deixando o diagrama de esquema intacto. A partir da geração de um diagrama de consulta, cada elemento pertencente a mesma se torna uma imagem, sobre a qual podem ser aplicadas as condições e projeções (a imagem pode ser expandida). Alguns fatores tais como a especificação explícita de junção sobre as imagens e a manipulação de operações lógicas ou quantificação utilizando *frames*, pode confundir um usuário leigo, tornando a especificação de uma consulta menos intuitiva do que em ConTOM. Contudo, um aspecto interessante em Graqula, que está ausente nas outras interfaces, é o mapeamento relacional da consulta, ou seja, durante a especificação gráfica da consulta, comandos SQL aparecem paralelamente, dentro da janela SQL.

5.3.4.2. Ambiente de Implementação

Com relação ao ambiente de implementação, são considerados os seguintes pontos: se as interfaces foram desenvolvidas a nível de protótipo ou produto e o hardware/software utilizado. O sistema SNAP foi desenvolvido em um *Symbolics* 3600, utilizando a ferramenta BOOGIE (Bell's Object-Oriented Graphics Interactive Executive) um pacote gráfico orientado a objeto. VQL roda sobre uma *workstation* Sun 3/60, o protótipo foi desenvolvido em *Smalltalk*-80. Muitas características gráficas que Graqula utiliza, aparecem em duas implementações E-R: o *browser* E-R AERIAL e o produto RMgraph. ConTOM é a única ferramenta que ainda não foi implementada, mas a ferramenta está projetada para *Sun/Sparc*, utilizando o *software* gráfico *Design/OA*.

5.3.4.3. Utilização de Gráficos

O último ponto central considera a **utilização de gráficos** na representação visual de esquemas, como paradigma de definição e acesso aos esquemas e na especificação de consultas. Um ponto adicional a considerar, é a classificação feita por *Foley* e *Dam* [Foley82] de tipos de *feedback* que um sistema interativo pode fornecer: léxico, sintático e semântico. O *feedback* semântico é a forma mais sofisticada de *feedback* e pode, por exemplo, indicar de uma maneira perceptível, que uma operação requisitada pelo usuário foi executada [Hull87]. As interfaces gráficas descritas, em geral, fornecem um bom *feedback* semântico, com a representação visual dos esquemas sendo dinamicamente modificadas assim que são manipuladas.

Todas as interfaces gráficas representam o esquema usando uma estrutura gráfica sob o modelo no qual elas se fundamentam e o usuário define diretamente a representação visual das mesmas e pode alterá-las, se for necessário (em ConTOM, existe a representação de diálogos na definição dos componentes de um esquema). Com relação ao acesso ao esquema, todas as interfaces percorrem diretamente sobre os componentes do esquema com o objetivo de focalizar áreas específicas de interesse. O estilo de interação básico sobre os esquemas, utilizado em todas as interfaces é a **manipulação direta** [Schneiderman83], o qual fornece um *feedback* semântico muito rico e eficiente para o usuário [Hull87].

Todas as interfaces utilizam recursos gráficos na especificação de uma consulta (ex: objetos gráficos, janelas). Em SNAP, ConTOM e Graqula, os componentes gráficos do esquema são manipulados diretamente para formar consultas, ou seja, o usuário identifica graficamente porções relevantes do esquema. Em VQL, o usário utiliza gráficos na combinação das primitivas visuais. Além dos recursos gráficos, as interfaces fornecem um suporte textual. Os predicados em SNAP são definidos textualmente enquanto que em ConTOM, o usuário utiliza, além de um suporte textual, menus *pop-up* na definição dos predicados.

5.3.4.4. Critérios Adicionais

- **Modelo de Dados**

Primeiramente, vamos apresentar o **modelo de dados** que cada interface gráfica suporta. QBE e Cupid suportam o modelo relacional, SNAP suporta um subconjunto do modelo semântico IFO, VQL suporta o modelo semântico SSONET, Graqula suporta tanto o modelo relacional quanto o modelo E-R e ConTOM suporta o modelo TOM.

- **Categoria de Classificação das Interfaces**

Mohan [Mohan93] classifica as linguagens de consulta dentro das seguintes categorias: linguagem de consulta textual, que já foi definida na seção anterior, linguagem baseada em "template", tal como QBE, linguagem de consulta gráfica, interface visual para BD's, linguagem visual de consulta e linguagem de consulta OO.

As **Linguagens de consultas gráficas** correspondem à formulação de consultas textuais sobre esquemas gráficos. Estas linguagens, portanto, não apresentam muito conteúdo visual. Linguagens como QBD* [Catarci90] pertencem a esta categoria. As **Interfaces visuais para BD's**, tais como Cupid [McDonald75], GUIDE [Wong82] e ISIS [Goldman85], fornecem a facilidade de acesso ao esquema (*browsing*) e tem como objetivo principal fornecer ao usuário a facilidade de localizar a informação, sem especificar a

descrição exata da informação ou onde ela está armazenada no BD. Contudo, elas não contém uma linguagem de consulta definida formalmente. As **Linguagens visuais de consulta (LVC)**, ao contrário, são caracterizadas pela existência de uma linguagem definida formalmente (com operações gráficas). Todas as interfaces que foram comparadas (SNAP, VQL, Graqula e ConTOM) se enquadram nesta categoria. O sistema SNAP, apesar de não apresentar uma sintaxe formal de consulta, se enqudra na categoria de uma linguagem visual, uma vez que sua especificação de consulta é feita através de uma sintaxe bidimensional, expressa visualmente. As **Linguagens de consulta OO** são linguagens que são especificadas por um modelo OO. Modelos OO tendem a ser complexos, e, consequentemente, as linguagens de consultas para estes modelos devem possuir um alto poder expressivo [Mohan93]. As linguagens OQL [Alasqur88] e ORION [Kim88] se enquadram dentro desta categoria, contudo, estas duas linguagens são completamente textuais.

- **Suporte ao esquema conceitual dividido em módulos**

Como já foi observado, somente as interfaces ConTOM e SNAP apresentam esta facilidade, ConTOM inclui a facilidade adicional de apresentar a dependência hierárquica entre os módulos, cuja estrutura é inerente ao esquema conceitual, não dependendo de uma consulta específica.

- **Suporte à consultas temporais**

ConTOM é a única interface que suporta consultas temporais dentro do mesmo ambiente interativo utilizado na formulação de consultas comuns. As linguagens temporais que existem atualmente especificam consultas temporais de uma forma completamente textual.

- **Linguagem interna**

Em algumas interfaces, existe um suporte interno de uma linguagem de consulta. SNAP e ConTOM não apresentam uma linguagem interna. Em VQL, as primitiva gráficas são mapeadas diretamente em primitivas textuais de uma linguagem de consulta tipo Prolog, baseada na lógica de predicados. Em Graqula, a linguagem interna é a linguagem SQL, o usuário inclusive pode visualizá-la.

- **Especificação formal para a sintaxe e semântica da consulta**

Em SNAP e Graqula, não existe uma especificação formal da consulta. Em VQL, a especificação formal é baseada na gramática BNF sobre as expressões válidas dentro da linguagem textual de predicados, enquanto que em ConTOM, a especificação formal é sobre as primitivas de consulta, baseada no trabalho de [Catarci93].

- **Mecanismos de Consulta**

Considerando os seguintes mecanismos que são utilizados na especificação de uma consulta: seleção de uma parte do BD, restrição, junção e projeção, vamos apresentar como estes mecanismos de consulta são representados interativamente em cada interface gráfica na tabela 5.1. VQL e ConTOM são as que apresentam homogeneidade de interação na representação dos mecanismos.

- **Especificação de consultas a nível de instância**

VQL é a única linguagem que apresenta esta característica, através dos elementos *dbInstances* e *dbInstanceLinks* que podem representar parte da informação dentro dos Ícones Classe e Ligação, respectivamente.

- **Consultas como operando de outras consultas**

Em ConTOM, esta facilidade é oferecida através do ER gerado em uma consulta e que pode ser salvo como um ECG. O ER aparece como uma estrutura modular, a partir da qual, o usuário pode gerar novas consultas. Em Graqula, o usuário pode salvar janelas de consulta como arquivos e um mecanismo de controle que faz parte do regulador, denominado regulador de execução, possibilita consultas invocarem outras consultas, e usuários e consultas passarem parâmetros para outras consultas. SNAP e VQL não apresentam esta facilidade.

- **Suporte à consultas recursivas**

ConTOM e SNAP não dão suporte à consultas recursivas. Em Graqula, a recursividade é efetivada através de aninhamento gráfico de *frames* (através dos frames que especifica a natureza e o escopo de uma operação lógica para elementos de consulta), que mostra o aninhamento gráfico de escopos (conjunto de operandos) e operações lógicas. Em VQL, a recursividade é expressa através do Ícone Ligação. Dois parâmetros adicionais precisam ser especificados dentro do Ícone Ligação, de forma a indicar a profundidade da recursão. O primeiro parâmetro é um inteiro especificando o nível de profundidade da recursão e o segundo parâmetro possui dois valores: o valor "*only*", indicando que apenas os nodos "folhas" da árvore, na profundidade de recursão especificada, devem ser recuperados; e o valor "*all*", indicando que todos os nodos da árvore devem ser recuperados.

- **Suporte à consultas com relacionamento reflexivo (cíclico)**

Um relacionamento reflexivo representa um relacionamento interagindo com ele mesmo (ex: *Empregado gerencia Empregado*). Todas as interfaces suportam relacionamento reflexivo. Em ConTOM,

particularmente, o usuário especifica uma consulta com relacionamento reflexivo, apenas selecionando os componentes gráficos na sequência: *classe1-relacionamento-classe1*.

- **Especificação de Quantificadores**

Em ConTOM e SNAP, não há uma especificação explícita para quantificadores na especificação de uma consulta. Em ConTOM, o quantificador implícito é o existencial - $\exists$ na semântica de uma consulta. Portanto, não será permitido especificar consultas que exigem o uso do quantificador universal - $\forall$. Em VQL, o quantificador universal é expresso explicitamente na parte inferior do Ícone Atributo. Em Graqula, a quantificação universal é expressa implicitamente através do *frame* implicação ou consequente, e a quantificação existencial é expressa implicitamente nas operações lógicas de conjunção e disjunção (maiores detalhes podem ser vistos em [Sockut93]).

- **Conjunção e Disjunção**

Existem dois níveis de conjunção/disjunção: conjunção/disjunção a nível de tupla ou instância, representado pelos operadores *and* e *or*, e a negação a nível de conjunto de instâncias, representado pelos operadores união ($\cup$) e intersecção ($\cap$). SNAP e VQL representam a conjunção/disjunção somente a nível de instância. Em Graqula e ConTOM, a conjunção/disjunção é representada nos dois níveis. Em ConTOM, a conjunção/disjunção é representada através das abstrações hierárquicas: generalização, que representa a união ($\cup$) e especialização múltipla, que representa a intersecção ($\cap$).

- **Representação da Negação**

Existem dois níveis de negação: negação a nível de tupla ou instância, representado pelo operador "$\neq$", e a negação a nível de conjunto de instâncias, representado pelo operador "~". SNAP e Graqula representam a negação somente a nível de instância. Em VQL e ConTOM, a negação é representada nos dois níveis. Em ConTOM, a negação a nível de conjunto de instâncias é representada através da especialização múltipla, utilizando o operador de diferença.

- **Operações agregadas**

Os operadores agregados (*sum, count, average,* etc) podem ser especificados nas interfaces VQL e Graqula. SNAP e ConTOM não suportam operadores agregados.

- **Atualização do BD**

A interface Graqula é a única que atende a este requisito. As atualizações (inserções, deleções e modificações são feitas através de uma coluna UPDATE, dentro de uma imagem expandida. A imagem pode ser de entidade ou relacionamento.

Para finalizar, a tabela 5.1. resume as interfaces gráficas dentro dos critérios que foram préviamente estabelecidos.

	SNAP	**VQL**	**Graqula**	**ConTOM**
FUNCIONALIDADE				
Visualização de Esquema				
Hierarquias de Abstração	✓	✓		✓
Níveis de Visualização	✓		✓	✓
Browsing de Esquema	Visual	Visual	Visual/Dir.	Visual
Definição de Esquema	✓		✓	✓
Metodologia de Projeto				FADO/*POKER*
IMPLEMENTAÇÃO				
Interface Gráfica	Protótipo	Protótipo	Protótipo	Protótipo
Hardware Utilizado	*Symbolics* 3600	Sun 3/60	IBM	*Sparc Station*
Software Utilizado	BOOGIE	*Smalltalk* 80	Aerial/RMgraph	*Design/OA*
UTILIZAÇÃO GRÁFICOS				
Repr. Visual de Esquema	Diagrama IFO	Diag. SSONET	Diagrama E-R	ECG TOM
Definição/Acesso Esquema	✓	✓	✓	✓
Especificação de Consulta	Man. Direta	Prim. Visual	Man. Direta	Man. Direta
CRITÉRIOS ADICIONAIS				
Modelo de Dados	IFO	SSONET	E-R	TOM
Categoria de Classificação	LVC	LVC	LVC	LVC
Consulta em Esq. Modular	✓			✓
Consultas Temporais				✓
Linguagem Interna		tipo Prolog	SQL	
Especificação Formal		✓		✓
Mecanismos de Consulta				
Seleção do BD	Comb. "reps"	Ícone Classe	Imagem E-R	OGC Seleção
Restrição	Restr. Nodo	Ícone Atributo	Imagem Exp.	OGC Espec.
Junção	Arco Comp =	Ícone Ligação	Junção *ad hoc*	OGC Seleção
Projeção	Destaque	Ícone Atributo	Destaque	OGC Agreg.

	SNAP	VQL	Graqula	ConTOM
Consulta a Nível Instância		✓		
Consulta como Operando			✓	✓
Consulta Recursiva		✓	✓	
Consulta com Rel. Reflexivo		✓	✓	✓
Quantificadores		✓	✓	
Conjunção/Disjunção	Instância	Instância	Inst/Conj. Inst.	Inst/Conj. Inst.
Negação	Instância	Inst/Conj. Inst.	Instância	Inst/Conj. Inst.
Operações Agregadas		✓	✓	
Atualização de BD			✓	

Tabela 5.1. Resumo Comparativo das Interface gráficas

Capítulo 6

Especificação da Ferramenta ConTOM

6.1. Introdução

Neste capítulo, proporemos um projeto de interface da ferramenta ConTOM, considerando três fatores que influenciam na escolha do estilo de interação do *software* a ser construído, segundo [Shneiderman87]: o objetivo do *software*, o tipo de usuário que utilizará o *software* e os recursos disponíveis para confeccioná-lo. Este projeto só foi parcialmente implementado, contudo, foram projetados todos os recursos necessários para o desenvolvimento futuro de uma ferramenta gráfica de consultas automatizada.

Um outro aspecto relevante, a ser considerado no projeto de uma interface gráfica, é a **ferramenta de especificação** sobre a qual a interface será desenvolvida. A maioria das interfaces gráficas que existem atualmente são baseadas em plataformas de desenvolvimento com recursos gráficos. A ferramenta de especificação corresponde a um *kit* de construção para prototipagem e implementação de ambientes gráficos, ou seja, é uma ferramenta que desenvolve ferramentas. Entre as inúmeras ferramentas de especificação existentes, o critério principal para a escolha das mesmas é que a ferramenta deva ser mais conveniente à funcionalidade e ao estilo de interação da interface que ela irá gerar. Outros critérios exigidos na escolha da ferramenta de especificação é a sua flexibilidade, interoperabilidade (multi-plataforma), ambiente horizontal (programação em alto nível), além da rapidez e eficiência. De acordo com estes critérios, o *software* proposto como ferramenta de especificação para o ambiente ConTOM é o *Design/OA (Open Architecture)* [MetaSoft89], que será descrito com mais detalhes no decorrer deste capítulo.

Este capítulo descreve a ferramenta ConTOM dentro dos seguintes aspectos: descrição da ferramenta *Design/OA,* descrição funcional da ferramenta ConTOM, as características principais do usuário que a utiliza e a descrição da interface. A descrição da interface envolve: método de entrada, estilos de diálogo, apresentação da informação (menus, *layout* de telas, destaque da informação), orientação em como utilizar a inteface e os dispositivos periféricos necessários.

6.2. *Design/OA (Open Architecture)*

O projeto de interface do ambiente ConTOM foi baseado sob o aplicativo ***Design/OA*** [MetaSoft89], uma ferramenta de especificação sofisticada, que fornece um ambiente **aberto** e **horizontal** para criação e manipulação de diagramas, gerando rapidamente produtos e protótipos gráficos, pois permite ao programador não se preocupar com detalhes de baixo nível referentes à interface gráfica (ex: gerenciamento de janelas, sistema de coordenadas) e sim com detalhes de alto nível relacionados à aplicação em si. É possível, através dele, desenvolver ambientes gráficos para CASE, CIM, CAD/CAE, gerar *front-ends* gráficos para BD's, redes de comunicação, etc, e implementar qualquer metodologia ou uma combinação delas (ex: modelo E-R, Redes de Petri, IDEF). Sua estrutura básica incorpora um vocabulário de objetos, conectores, textos, hipertexto e hierarquia de diagramas.

No *Design/OA*, pode-se determinar um estilo particular de operação: determinar quais os objetos que serão reconhecidos pela aplicação (analisador léxico); determinar as regras que governam os relacionamentos entre os objetos (analisador sintático) e definir o significado dos objetos relacionados (analisador semântico). Adicionalmente, as aplicações em *Design/OA* são portáveis para os ambientes *MS-Windows, Macintosh Tollbox* e *X-Windows*.

6.2.1. A Estrutura de uma Aplicação *Design/OA*

Aplicações feitas no *Design/OA* possuem três componentes:

1. O ***kernel***, que corresponde às rotinas e estruturas de dados que compõem o editor gráfico *Metadesign* [MetaSoft93], que será visto a seguir.

2. A **interface *Design/OA***, que corresponde a uma biblioteca de funções que acessa o *kernel*. Esta biblioteca suporta criação, manipulação e *display* de diagramas, menus, *dialog boxes*, etc.

3. **O módulo de desenvolvimento,** refere-se à própria aplicação, codificada em C e que faz chamadas a interface ou, opcionalmente, ao ambiente sob o *Design* (*MS-Windows, Macintosh* ou *X-Windows)*, controlando a interação entre o usuário final e o *Kernel*.

A figura 6.1 mostra os componentes em níveis para o funcionamento de uma aplicação *Design/OA* (*RID* significa Rotinas de Interface de Desenvolvimento, X representa a plataforma *X-Windows* e OS representa o sistema operacional *Sun/OS*).

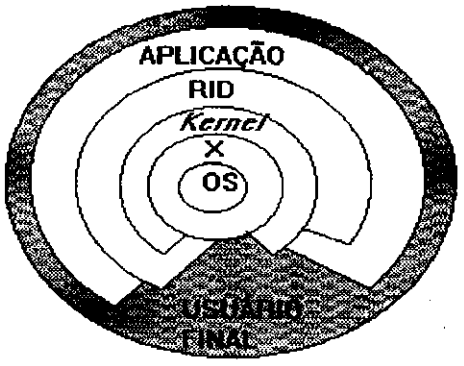


Figura 6.1. Níveis de Funcionalidade de uma Aplicação Design/OA

6.2.2. O *Kernel Design/OA*

O *Kernel* contém rotinas e estrutura de dados em baixo nível que compõem o ***Metadesign***, uma ferramenta para manipulação de gráficos orientados a objeto e texto. O *Design/OA* é a versão *Metadesign* com arquitetura aberta, ou seja, ele extende e modifica o *Metadesign* conforme o contexto de uma aplicação específica. Equivalente a muitos editores gráficos, *Metadesign* permite a criação de objetos gráficos, com formatos variados (círculos, retângulos, etc), com *display* de atributos variados (*fill, line)*, assim como pode manipular os objetos individualmente ou em grupos (movimentar, diminuir, etc), além da utilização de *palettes* e importação de outros gráficos. Contudo, *Metadesign* possui muitas outras características poderosas, entre as quais: a lógica de grafos conectados: conectores são automaticamente redesenhados quando os seus nodos são movimentados; hierarquias de objetos e diagramas; e integração de textos, onde blocos de texto podem ser associados aos nodos e conectores, ou manipulados individualmente (blocos de texto podem ser compostos hierarquicamente, formando hipertextos).

Eis os seguintes conceitos básicos do *Metadesign* e a correspondência aos conceitos básicos de ConTOM:

- **Página**: um documento *Metadesign* consiste de uma ou mais páginas sobre as quais são criados os objetos gráficos e texto. Cada página é vista através de uma janela com *scroll-bar*. Páginas e objetos podem ser ligados hierarquicamente ou não. Um documento equivale a um ECG e uma página equivale a um módulo, em ConTOM.

- **Objetos Gráficos**: representam os nodos e os conectores. Nodos, adicionalmente, podem ser classificados como regiões, que são subordinados a outros nodos. Os nodos equivalem às classes e abstrações e os conectores são os relacionamentos, em ConTOM.

- **Texto**: texto em *Metadesign* é associado a um objeto gráfico. Pode-se ter acesso a todos os fontes, tamanhos e estilos. Corresponde às denominações dos componentes gráficos de um ECG.

- **Relacionamentos Lógicos**:pode-se relacionar níveis lógicos de informação através de relacionamentos lógicos que reduzem a complexidade gráfica de uma página, movendo detalhes para uma subpágina, através da existência de um nodo *coarse* na página *pai*, que vai representar o detalhe movido para a subpágina ou página *filha*. Os relacionamentos lógicos são adequados para representar as abstrações em um ECG modular.

6.2.3. A Interface *Design/OA*

Design/OA fornece uma interface de alto nível para ter acesso às rotinas e estruturas de dados do *Metadesign*, ou seja, a interface representa a ligação entre o *Kernel* e o módulo de desenvolvimento. *Metadesign* trabalha para processar cada evento que ocorre em seu ambiente (tal como uma ação do usuário) e traduz este evento em chamadas de função que respondem apropriadamente. Utilizando o *Design/OA*, pode-se definir novas ações do usuário. A tradução destas novas ações pode ser construída a partir de rotinas dentro da biblioteca de interfaces, incluindo funções que:

- Criam, acessam ou modificam a estrutura do diagrama.
- Lêem ou modificam os atributos das páginas, nodos, regiões, conectores ou texto.
- Controlam a seleção de objetos gráficos, páginas ou grupos.
- Criam e manipulam menus.
- Criam e manipulam *dialog boxes*.
- Criam e manipulam dados do usuário.

Em suma, eis os principais recursos do *Design/OA* que foram utilizados em ConTOM:

- Diversos objetos gráficos com funções pré-definidas de criação, seleção, movimento, remoção e modificação de características visuais.
- Decomposição hierárquica automática dos diagramas.
- Objetos conectados logicamente com reorganização automática do diagrama, caso haja algum movimento de algum objeto.
- Diálogos em formato de formulários padronizados para obter informações dos objetos gráficos.
- Menus pré-definidos para manipular e construir diagramas.

6.3. Funcionalidade da Ferramenta ConTOM

A ferramenta ConTOM é uma ferramenta que auxilia o usuário:

1. Na representação estrutural das classes, relacionamentos e abstrações do modelo TOM, através da construção de um ECG. A ferramenta acompanha o usuário na entrada das informações sobre os componentes do ECG através de formulários baseados nos formulários padronizados da metodologia FADO. As informações obtidas são colocadas em um único dicionário e o ECG gerado fará parte de uma biblioteca de esquemas.

2. Na especificação de consultas a uma base de dados, através da manipulação direta sobre um ECG. As informações detalhadas sobre cada componente podem ser obtidas através do acesso ao dicionário.

As funções da ferramenta ConTOM são:

- Dar suporte à metodologia FADO na construção do diagrama estático (ECG), fornecendo controle das informações sobre um único dicionário de dados.
- Obter as informações contidas dentro do dicionário de dados.
- Acompanhar o usuário durante o processo de especificação de uma consulta, desde o acesso ao ECG, até o fornecimento do resultado da consulta.
- Atender ao usuário fornecendo uma interface amigável com utilização de *mouse*, janelas, ícones, menus *pop-up* e *pull-down*, incluindo um menu de ajuda *on-line*, que explicará o funcionamento dos OGC's através de exemplos, durante a especificação de uma consulta.

6.4. Característica do Usuário

Existem duas categorias de usuário que utilizam a ferramenta ConTOM: o **projetista**, que define o ECG e o **usuário final**, que acessa os dados através dos OGC's sobre o ECG. Vamos considerar somente o perfil do usuário final. O perfil do projetista pode ser visto em [Furtado93b]. A ferramenta ConTOM foi projetada para um usuário final com as seguintes características:

- Habilidade Técnica: Nocões de BD's.
- Não precisa ter experiência com computadores.
- Frequência de utilização: ocasional.

- Experiência com o sistema: a interface é projetada para usuários noviços, que não tem conhecimento sintático sobre o sistema e pequeno conhecimento semântico sobre o computador. O ambiente proporcionará uma interação adequada na especificação de consultas, através dos seguintes recursos:
  1. Opção para formular consulta graficamente.
  2. *Feedback* imediato a cada tarefa do usuário.

3. Mensagens indicando o estado corrente.
4. Mensagens de alerta quando algum erro surgir e o destaque sobre o erro.
5. Ajuda *on-line.*
6. Menus com uma terminologia familiar.

Os usuários noviços, desta forma, terão acesso às informações de como manipular o ambiente, sem precisar memorizá-las, além de possuírem tempo disponível para realizar uma função, interagindo graficamente com o ambiente.

- Conhecimento Conceitual: o usuário deve ter noções sobre:
  1. Expressões Booleanas (*or, and, not*)
  2. Teoria de Conjunto (união, intersecção, diferença)
  3. Esquemas Conceituais de BD's (classes, atributos, abstrações, aspectos temporais, etc).

- Conhecimento do Método: os usuários devem aprender o significado dos componentes de um diagrama de esquema conceitual, para poder saber manipulá-los.
- Apresentar habilidade motora: manuseio do *mouse* sobre os objetos, itens de menu e janelas.
- Não precisar de treinamento prévio.
- Ser Curioso e Persistente.
- Apresentar boa percepção.

6.5. Descrição da Interface da Ferramenta ConTOM

6.5.1. Estilos de Diálogos

1. Formulários:

O formulário é o estilo de diálogo mais apropriado para a tarefa de entrada de dados, mais especificamente, na obtenção de informações dos objetos gráficos componentes de um ECG. A entrada das informações é feita em *dialog boxes.* Cada item de informação é validado automaticamente ao preencher um campo do formulário.

2. Menus:

Os menus possuem como principal característica a não necessidade de memorização de teclas de função ou comandos a serem executados. Portanto, os menus de ConTOM, por apresentarem itens com terminologia familiar (o nome do item está estreitamente associado com a função que ele exerce),

reduzem consideravelmente os erros, desde que não se pode fugir dos itens apresentados. Para usuários experientes ou com uma frequência de utilização regular no ambiente, os menus são também adequados, pois cada menu apresenta poucos itens, não tornando seu uso cansativo ou tedioso.

3. Manipulação Direta:

A escolha da manipulação direta na ferramenta ConTOM é devido a este estilo de interação ser bastante apropriada a ambientes gráficos. Existem outros motivos que justificam a manipulação direta:

- Perfeitamente adequado para as duas categorias de usuário que manipularão a interface (projetista e usuário final).
- Adequado ao tipo de aplicação existente na ferramenta, mais especificamente, a geração de um ECG e a formulação da consulta no modo gráfico, através da manipulação sobre os objetos componentes do ECG. Acrescentado a esta manipulação, existem as facilidades do *feedback* gráfico imediato a cada ação direta do usuário e a operação de reversibilidade com o mecanismo de *undo*.
- A manipulação direta no ambiente, é feita sobre todas as funções fornecidas pela interface, tais como: manipulação dos menus, janelas, ícones, objetos gráficos, etc, sempre apresentando o estado corrente da formulação de uma consulta através da visibilidade dos objetos e as ações sobre eles.
- A manipulação direta é o estilo de interação básico dentro da ferramenta de desenvolvimento *Design/OA.*.

Em suma, a manipulação direta é muito atraente, pois sua forma de interação é compreensível, natural, rápida e estimulante. Reduz a ansiedade dos usuários se as ações são simples, reversíveis e de fácil retenção, o que facilita a aprendizagem [Shneiderman83].

6.5.2. Entrada no Ambiente ConTOM

Vamos considerar o funcionamento do ambiente ConTOM em uma estação SUN. Após a entrada no ambiente SUN (através dos comandos *login* e *password*), o usuário deve digitar a palavra "ConTOM". Após este procedimento, aparecerá uma tela que corresponde ao menu principal que contém as funções iniciais do ambiente e permite continuar ou sair de dentro do mesmo, como ilustrado na figura 6.2.

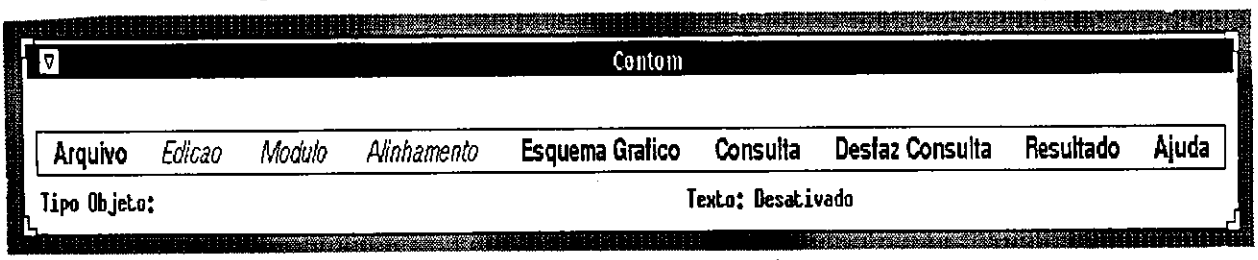


Figura 6.2. Menu Principal de ConTOM

6.5.3. Funções de Interface da Ferramenta ConTOM

Vamos apresentar as funções que levam em consideração a ferramenta *Design/OA* e indicam como o usuário cria, acessa e manipula um ECG durante uma consulta e como o sistema fornece o resultado das informações, além do suporte de ajuda ao usuário, detecção/correção de erros e finalização das operações.

6.5.3.1. Apresentação das Informações

As informações são apresentadas através de menus, janelas e diagramas. Os *layouts* das telas são ilustrados na figura 6.3. O *layout* principal (*layout* 1) contém as seguintes informações:

1. **Área de Título**: contém o nome da aplicação (ConTOM).
2. **Área de Menu**: contém um menu *bar*.
3. **Área de Mensagens**: contém os campos *tipo* e *Módulo%* e todas as mensagens do sistema, com exceção das mensagens de erro.
4. *: ativa um menu *pop-up* com funções de manipular janelas (ex: transformar a janela em ícone, aumentar, reduzir, etc).

O *layout* 2 representa o *layout* de um módulo de um ECG . A **área de trabalho** corresponde a uma janela com *scroll bar*, sobre a qual é mostrado um ECG.

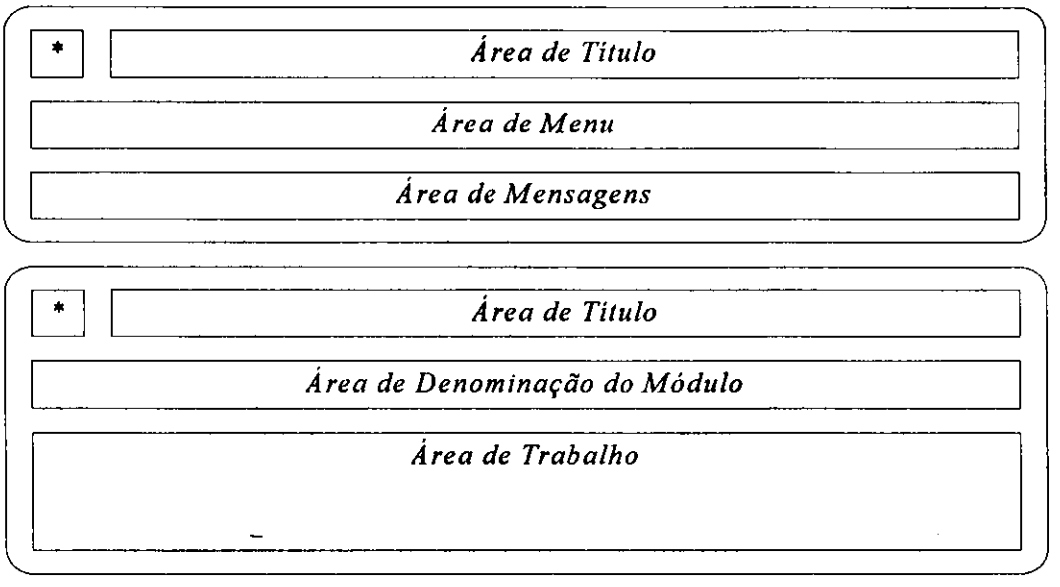


Figura 6.3. ***Layout*** **de telas do Ambiente ConTOM**

Os menus contém as funções do ambiente ConTOM e estão organizadas hierarquicamente em dois níveis. No menu principal, representado como um menu *bar*, as funções estão em um nível de abstração mais alto. A seleção de um item no menu principal ativará um menu *pull-down* contendo um agrupamento lógico de

sub-itens referentes àquele item, um exemplo de menu *pull-down* ativado pelo item *Arquivo* é ilustrado na figura 6.4.

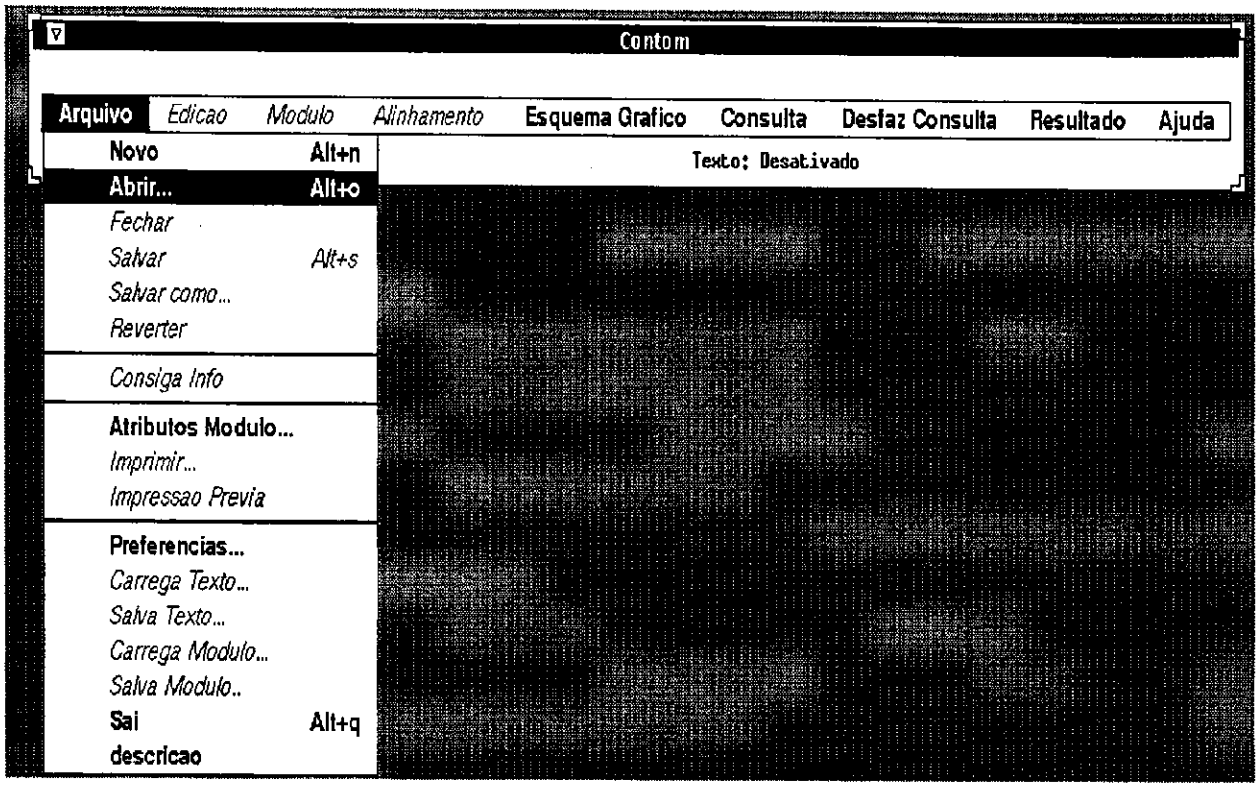


Figura 6.4. Exemplo de um Menu *Pull-Down* Ativo

Cada módulo é visto através de uma janela com *scroll bar*, como ilustra a figura 6.5. As janelas podem ser selecionadas, movimentadas, aumentadas, reduzidas, transformadas em ícone, etc. Está disponível na área de mensagens, os seguintes campos: *tipo* e *Módulo%* . Esses campos tem comportamentos dinâmicos, isto é, as informações variam de acordo com o que estivermos fazendo. Por exemplo, no caso da edição ou seleção de um objeto gráfico, é mostrado no campo *tipo*, se esse objeto é uma classe, um relacionamento ou uma abstração qualquer. O campo *Módulo%* contém o percentual de espaço de um diagrama dentro daquele módulo, esta informação é útil para percebermos a existência de *zooms*, porque para apresentarmos um diagrama de forma mais clara, podemos visualizar partes de um diagrama de forma mais detalhada. Todos os módulos são nomeados e numerados, para facilitar o manuseio dos mesmos, além da ferramenta fornecer a estrutura hierárquica dos módulos de um ECG modular (ver capítulo III, figura 3.8).

Vamos retomar ao conceito de relacionamento lógico dentro do *Metadesign*, definido na seção 6.2.2. Em um ECG modular, os diagramas dentro dos módulos (páginas) são organizados hierarquicamente, onde cada nível do diagrama está disposto em uma único módulo (ver seção 3.3.3.2). Existe o módulo *pai* e o módulo *filho*, onde este último é a explosão das idéias genéricas contidas no módulo *pai* associado. Pode-se acessar o módulo *filho* a partir do módulo *pai*, através do *"click"* sobre este último módulo. A figura 6.5 mostra como ConTOM apresenta os diagramas em níveis de abstrações diferentes. Na figura estamos

considerando a visualização de três módulos do exemplo de modularização de um ECG (ver cap. III, seção 3.3.3.2).

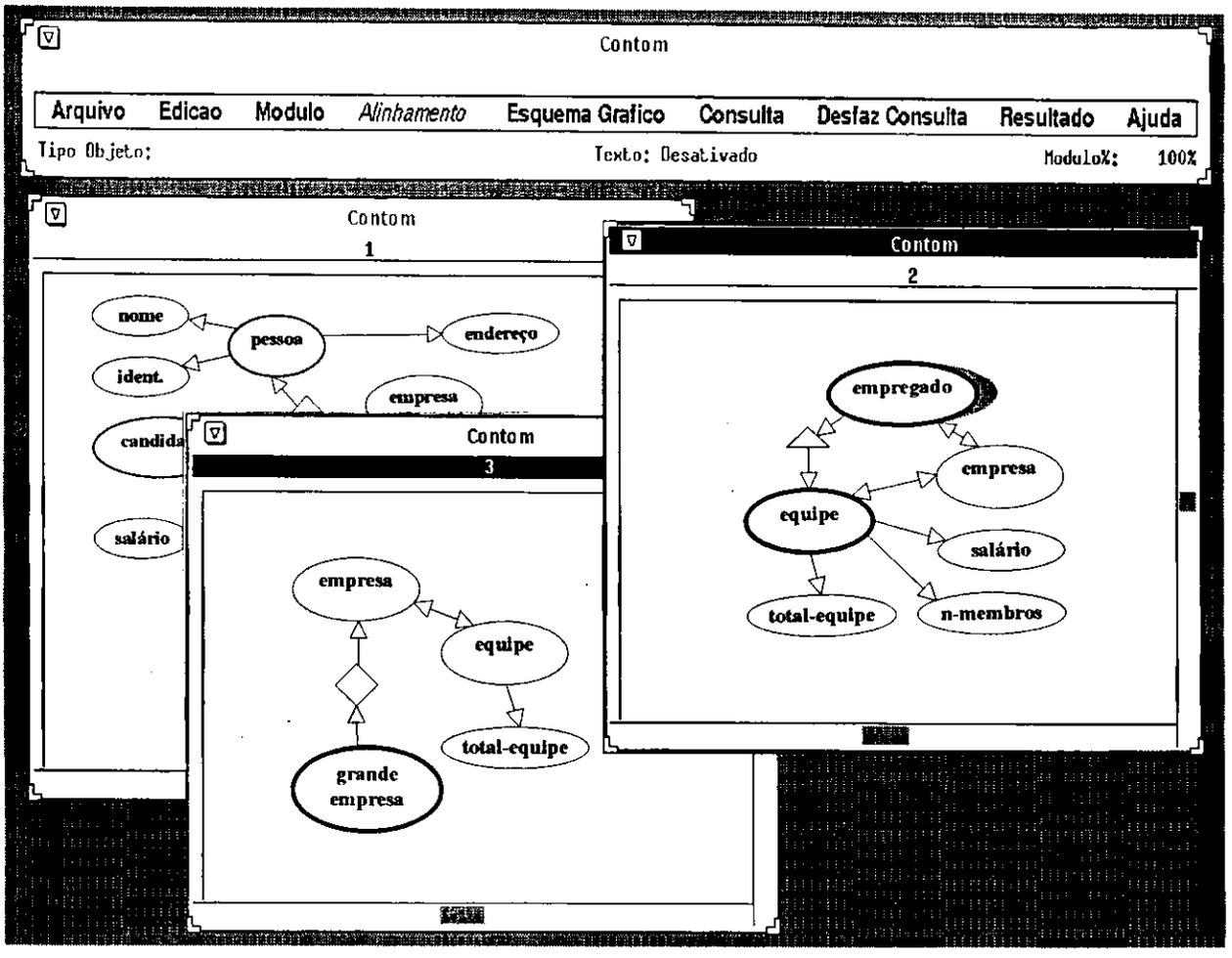


Figura 6.5. Exemplo de Visualização de Três Módulos

6.5.3.2. Entrada das Informações

A entrada das informações é feita, durante a criação de um ECG, através dos *dialog boxes*, que representam as informações de cada objeto gráfico de um ECG e devem seguir o mesmo formato dos modelos dos formulários desses objetos, definidos na metodologia FADO [Furtado93a]. A entrada dessas informações é feita pela digitação direta dentro dos formulários. Por exemplo, na figura 6.6, após a seleção da opção *Novo* do menu *Arquivo*, indicando a criação de um novo ECG, e logo a seguir, a seleção da opção *Classe Temporal* do menu *Esquema Gráfico*, e o desenho de uma classe temporal feito pelo usuário, aparecerá um *dialog box*, mais especificamente, um *edit box*, com o formulário da classe. Após o preenchimento do formulário, o nome da classe temporal é colocado dentro da figura representativa e outras informações vão para o dicionário de dados. Se a classe já foi desenhada em um outro módulo, não é necessário preencher as informações novamente. O acesso aos formulários pode ser feito através do *"click" duplo* sobre o objeto gráfico.

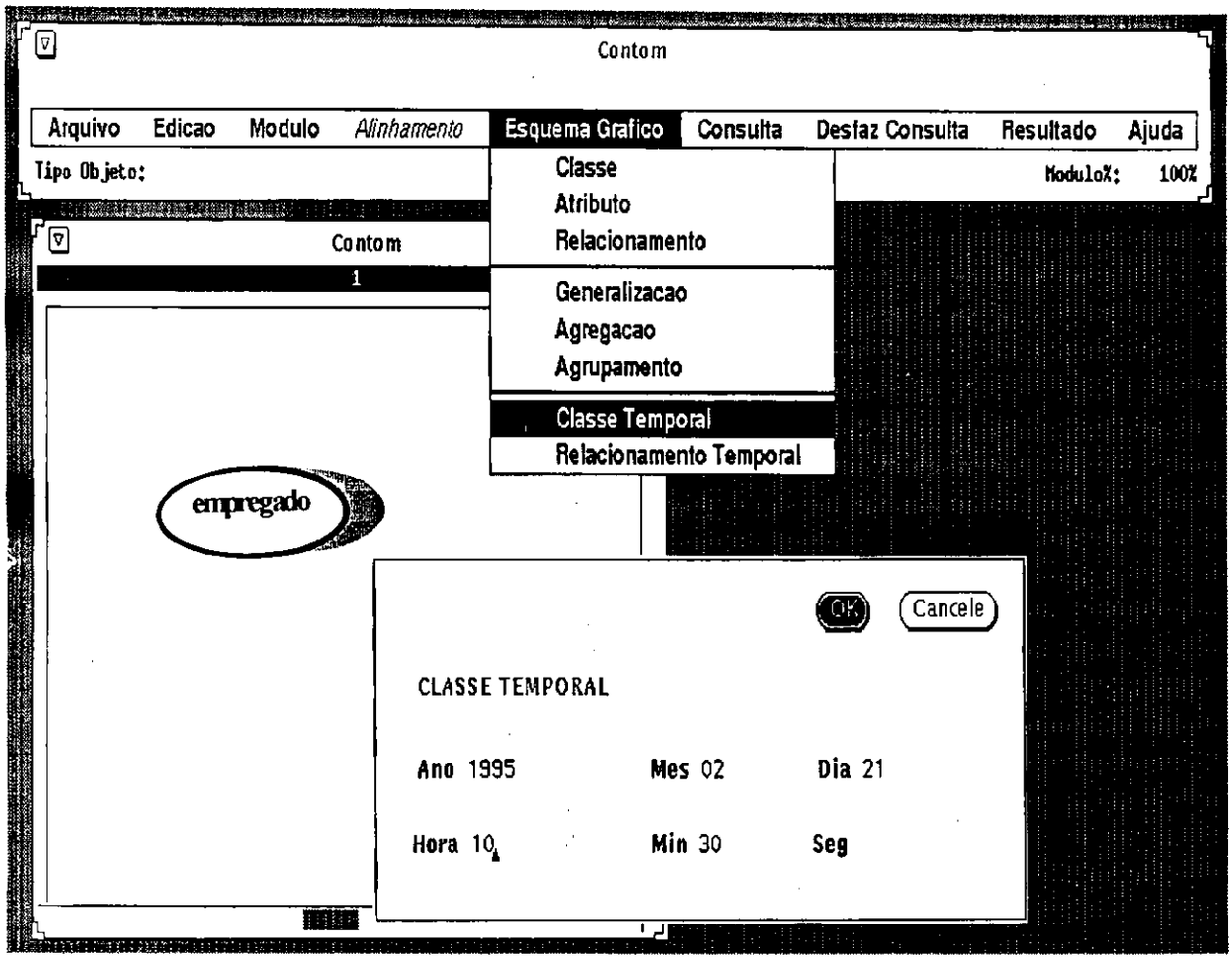


Figura 6.6. Exemplo de Criação de uma Classe Temporal

Adicionalmente, existem *dialog boxes* para as seguintes funções:

- Mensagens de erro, na forma de um *alert box*.
- Cancelamento de uma Execução.
- Indagações que esperam alguma tomada do usuário.
- Coleta de informações em formato de *edit box*.
- Elementos de Controle interagindo com o usuário na forma de *radio buttons*.

Um exemplo de *dialog box* é ilustrado na figura 6.7, que representa a abertura de um ECG.

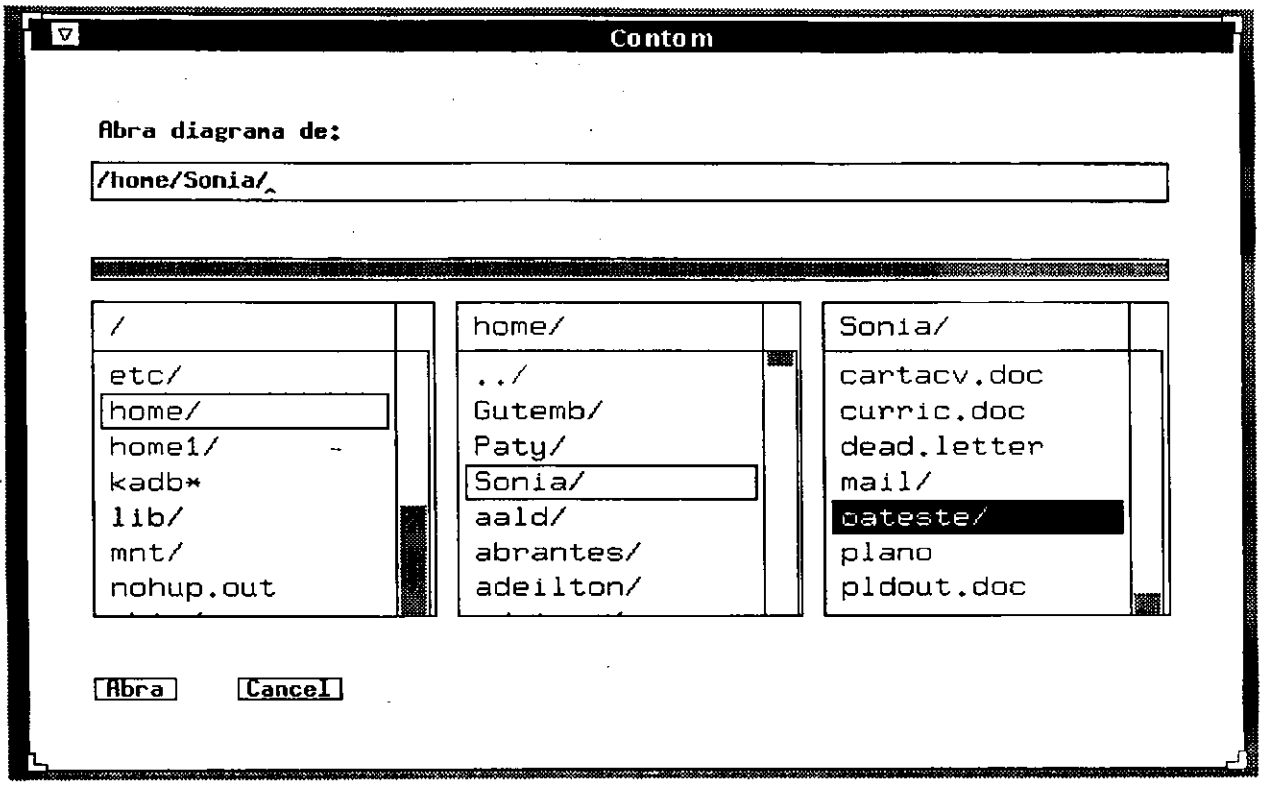


Figura 6.7. Abertura de um ECG

6.5.3.3. Seleção das Opções

Quando trabalhamos com o *mouse*, a seleção das opções pode ser feita diretamente *"clickando"* o botão sobre a opção desejada. As opcões compreendem itens de menu, ativação de uma janela, seleção de um objeto gráfico, etc (as opções ativadas ficam em vídeo reverso).

Vamos considerar a manipulação dos OGC's. Os OGC's aparecem como itens do menu principal *Consulta*. Na seleção de um OGC específico, o usuário faz a seleção de um ou mais objetos gráficos. No momento em que um objeto gráfico é selecionado, ele aparece em destaque no módulo. O usuário só pode selecionar os objetos gráficos perante a restrição da lógica do OGC escolhido; a restrição corresponde às regras descritas no capítulo IV. Qualquer ação do usuário que viole estas regras, ConTOM apresentará uma mensagem de alerta. Por exemplo, o usuário acessou um ECG não modular e quer consultá-lo, como ilustra a figura 6.8. No menu de consultas o usuário escolhe uma das opções mostradas com o *"click"* do *mouse*. Note que a único OGC disponível é o OGC *seleção*. As opções disponíveis vão estar em *highlight*, caso contrário, elas dependem de algum evento ou condição para se tornarem disponíveis (no exemplo, os outros OGC's vão estar em *highlight* quando a operação de seleção estiver concluída).

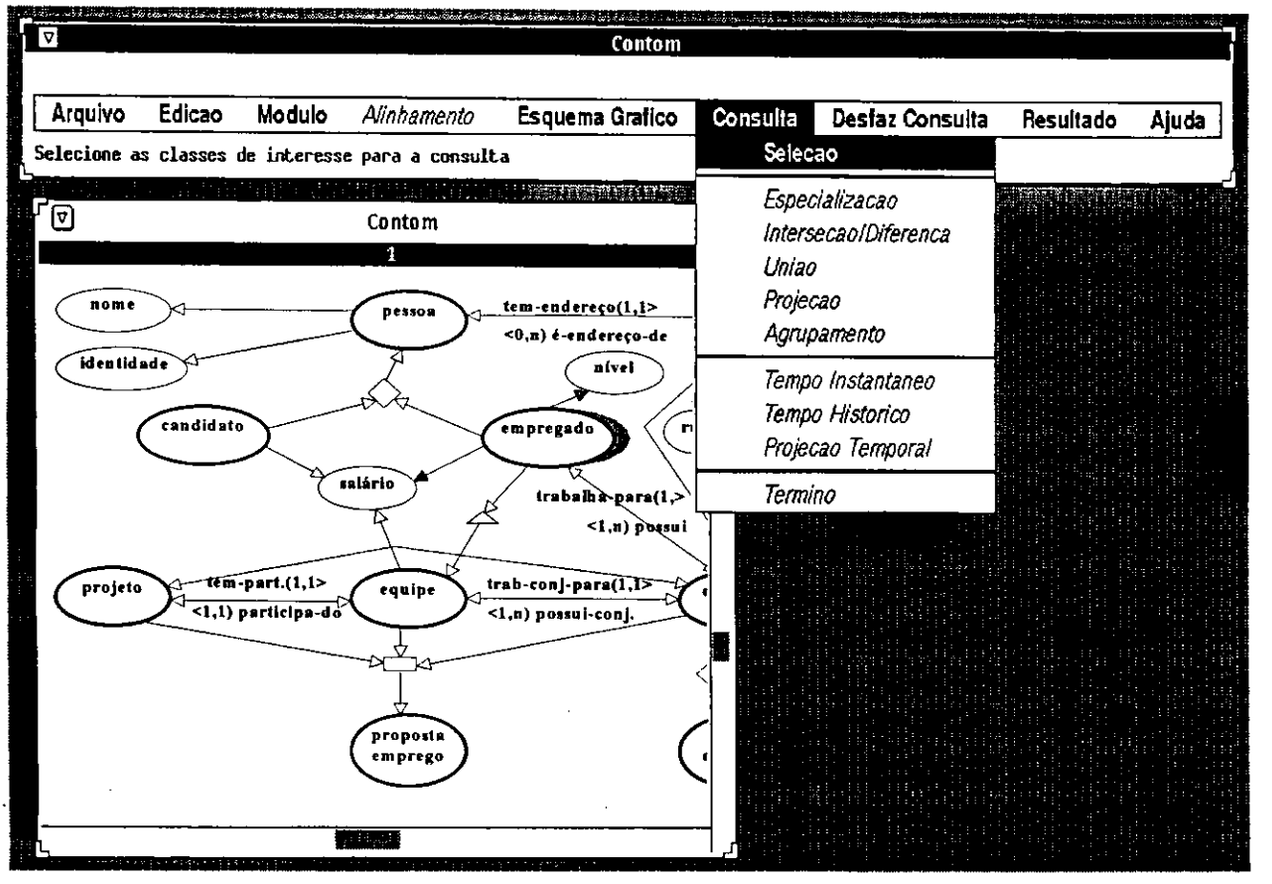


Figura 6.8. OGC *Seleção* Ativo

Na seleção do operador de *especialização*, aparecerão *dialog boxes* e menus *pop-up* para para a construção de um predicado. Um menu *pop-up* aparece na escolha do operador de comparação e na escolha do operando direito, *dialog boxes* aparecem na entrada de uma constante como valor de um atributo, na edição de expressões booleanas (para predicado complexos), na denominação da nova classe derivada, etc, como ilustra a figura 6.9. De forma análoga são construídos os predicados de outros operadores de consulta.

6.5.3.5. Saída da Informação

A saída de uma informação em uma especificação de consultas corresponde ao resultado da mesma. O menu principal *Resultado* apresenta os seguintes itens: *Esquema de Resposta* (ER), onde o ambiente fornece a representação gráfica do ER gerado (o ER aparece em uma estrutura modular) após a formulação da consulta e *Execução*, onde o ambiente fornece o resultado da consulta. O ambiente permite salvar o ER gerado na biblioteca de ECG's através da opção *Salvar* do Menu *Arquivo*, podendo ser reutilizado futuramente.

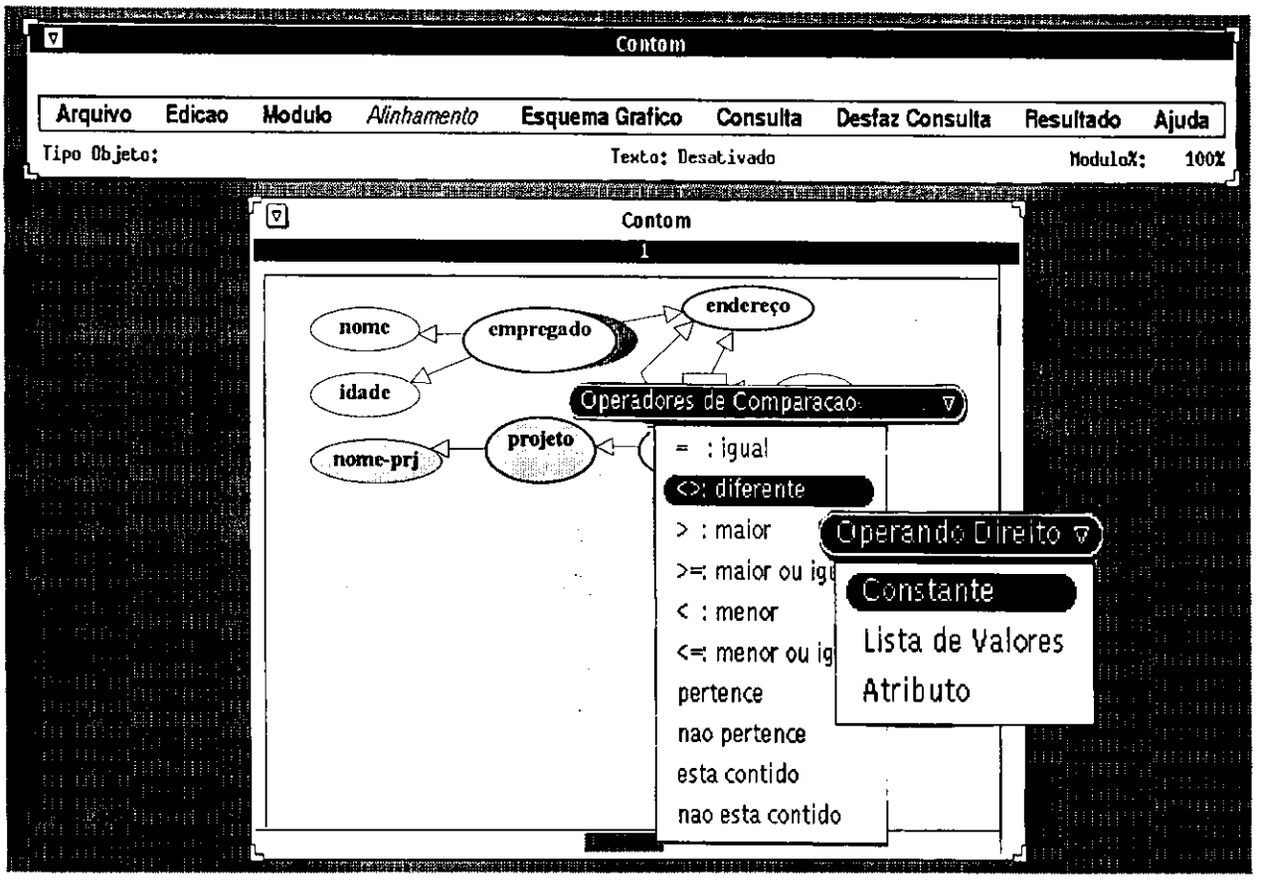


Figura 6.9. OGC *Especialização* Ativo

6.5.3.6. Retorno de um Menu à Função que o Chamou

Caso o usuário não queira selecionar nenhum item de um menu *pull-down* ou *pop-up* que estaeja ativado, o usuário retorna à função que o chamou através do *"click"* sobre a mesma ou outra função qualquer.

6.5.3.7. Orientação no Uso da Interface

Existem diversas maneiras de orientar o usuário ou projetista dentro da ferramenta ConTOM. A primeira maneira é através das mensagens de orientação ao usuário em como proceder após alguma ação que ele realizou. Estas mensagens ficam localizadas na Área de Mensagens (ver *layout* 1 da figura 6.3). Um exemplo que utiliza mensagens é ilustrado na figura 6.8, quando o usuário ativa a seleção, aparece a mensagem "*selecione as classes de interesse para a consulta*".

A segunda maneira é através dos campos *Tipo* e *Nome do Módulo* (ver figura 6.5) que indica ao usuário onde está e o que faz. Para saber onde ele irá, a ferramenta ConTOM fornece um menu de *Ajuda* que se encontra no menu principal e apresenta as seguintes funções:

1. Descrição funcional do ambiente através da apresentação de todas as funções que a compõem. Cada função vem com a sua descrição genérica e como deve acessá-la, através da ilustração de um exemplo correspondente.
2. Descrição a nível semântico dos conceitos utilizados no modelo TOM e como cada conceito pode ser representado em um ECG, mostrando exemplos de esquemas conceituais.
3. Passos a serem seguidos na manipulação de um ECG para uma consulta, através dos OGC's. Exemplos ilustrativos de consultas simples são mostrados.

Se o usuário quiser saber onde está, dentro do ECG, ele pode visualizar a estrutura dos módulos do ECG corrente, através da seleção do item *Estrutura* do menu *Módulo* (o módulo corrente fica em destaque), como ilustra a figura 6.10.

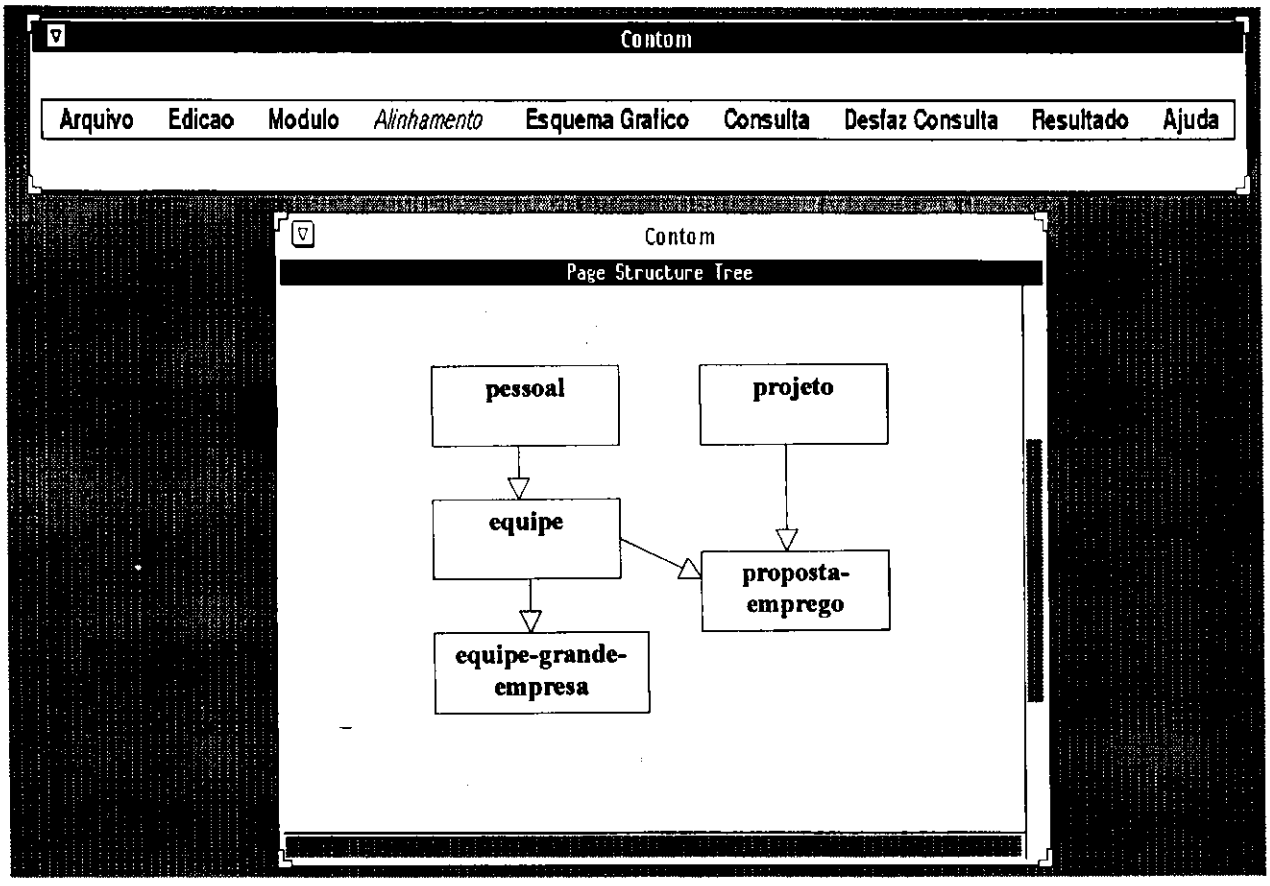


Figura 6.10. Exemplo de uma Estrutura Modular

6.5.3.8. Detecção e Recuperação de Erros

Outra forma de orientar o usuário é na detecção e correção de erros. ConTOM previne erros durante a construção e manipulação de um ECG. Por exemplo, ele não permite: a conexão entre dois nodos que representam abstrações, a criação de um objeto gráfico já existente, a utilização de um OGC temporal sobre uma classe ou relacionamento não temporal, etc. Tais erros nem chegam a ser efetivados pois são bloqueados pelo sistema. As advertências vem representadas em *Alert Boxes*.

As operações consideradas indevidas pelo usuário, podem ser canceladas através do mecanismo *Undo*. Este mecanismo está presente em dois menus: no menu *Edição*, para desfazer a última ação do usuário, e dentro do menu *Consulta*, para modificar uma manipulação de um OGC, por exemplo, o usuário quer modificar a restrição de uma classe, desta forma, ele seleciona, em sequência, a opção *Undo*, o operador *Especialização* e o atributo sobre o qual foi especificado a restrição, como ilustra a figura 6.11:

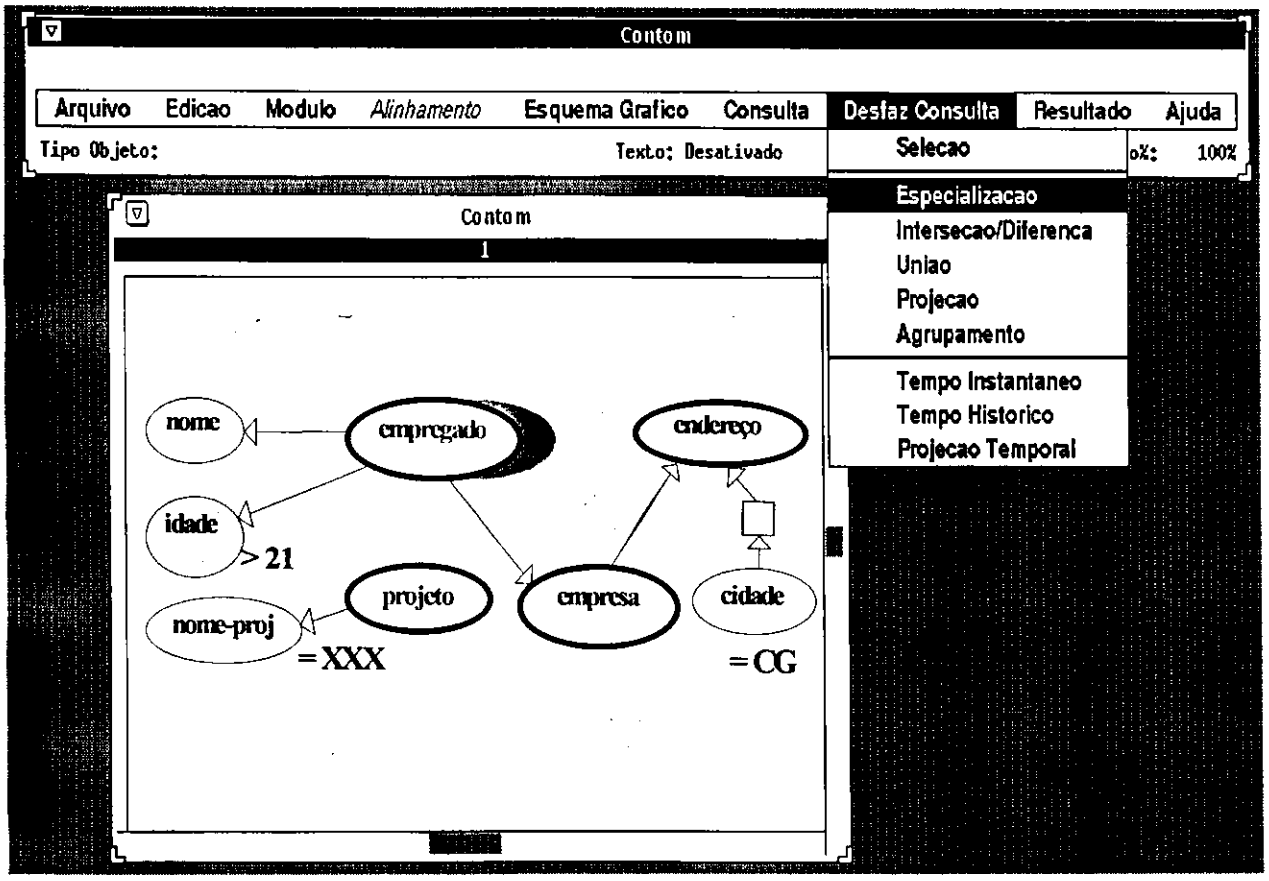


Figura 6.11. Mecanismo *Undo* Ativo

6.5.3.9. Documentação

Existem dois tipos de documentação dentro da ferramenta ConTOM: a ajuda *on-line*, descrita na seção anterior, e a documentação textual apresentada através de um manual. O manual deve conter as seguintes informações:

1. Data em que foi desenvolvida.
2. Em que versão se encontra.
3. Instituição que a desenvolveu.
4. Índice.
5. Dados de Implementação.
6. Breve resumo do funcionamento da ferramenta.

7. Procedimentos passo a passo de como manusear a ferramenta.
8. Exemplos ilustrativos.
9. Tabela de mensagens de erro e a explicação de cada mensagem.
10. Glossário.

6.5.3.10. Finalização das Operações

A qualquer momento, o usuário pode sair do ponto em que está através do *"click"* sobre o quadrado situado no canto superior esquerdo das janelas (a janela se transforma em ícone). A saída do ambiente e retorno ao sistema operacional é feito através da seleção *Sai* do menu *Arquivo*.

6.5.4. Dispositivos Periféricos

Os dispositivos de entrada utilizados são: o teclado, que serve para acessar funções através de teclas programáveis e é necessário na formulação de predicados; e o *mouse*, que é totalmente imprescindível para o ambiente, desde que o usuário interage com o mesmo através da manipulação direta sobre os objetos gráficos e itens de menus. Os dispositivos de saída são o monitor de vídeo monocromático e com tela plana ou arquivo que conterá os resultado; a escolha do monitor ser monocromática é devido à limitação da ferramenta *Design/OA* não apresentar um ambiente gráfico colorido.

6.5.5. Funções de Apoio à Ferramenta ConTOM

Esta seção mostra as funções disponíveis na interface do aplicativo *Design/OA* e que fazem parte da ferramenta ConTOM. Estas funções são de apoio à construção e manipulação dos módulos e objetos gráficos de um ECG. Maiores detalhes sobre estas funções podem ser vistas em [MetaSoft89].

6.5.5.1. Edição dos Objetos Gráficos

Corresponde ao menu *Edit* do *Design/OA*. É na edição que o projetista constrói ou modifica um ECG. Através dele podemos selecionar, remover e copiar objetos gráficos. Esta função é desativada quando o menu de *Consulta* é ativado, ou seja, não será permitido a edição dos objetos durante a formulação de uma consulta. A única forma de manipulá-los é através do *"click"* sobre os objetos e a escolha de um OGC específico.

6.5.5.2. Manipulação de Módulos

Corresponde ao menu *Page* do *Design/OA.*, através dele podemos definir atributos de um módulo, como ilustrado na figura 6.12. Um módulo pode ser mostrado, aumentado *(zoom)*, diminuído, criado, destruído e inicializado (estas três ultimas funções só podem ser feitas pelo projetista). Para navegarmos entre os módulos, devemos escolher os itens *Módulo Pai* e *Módulo Filho*. O item *Estrutura* permite-nos ver todos os relacionamentos hierárquicos entre os módulos. Quando o usuário estiver trabalhando com vários módulos, ele poderá tranformar alguns em ícone no final da tela. Para trazê-los de volta ao tamanho normal, é só *"clickar"* sobre ícone.

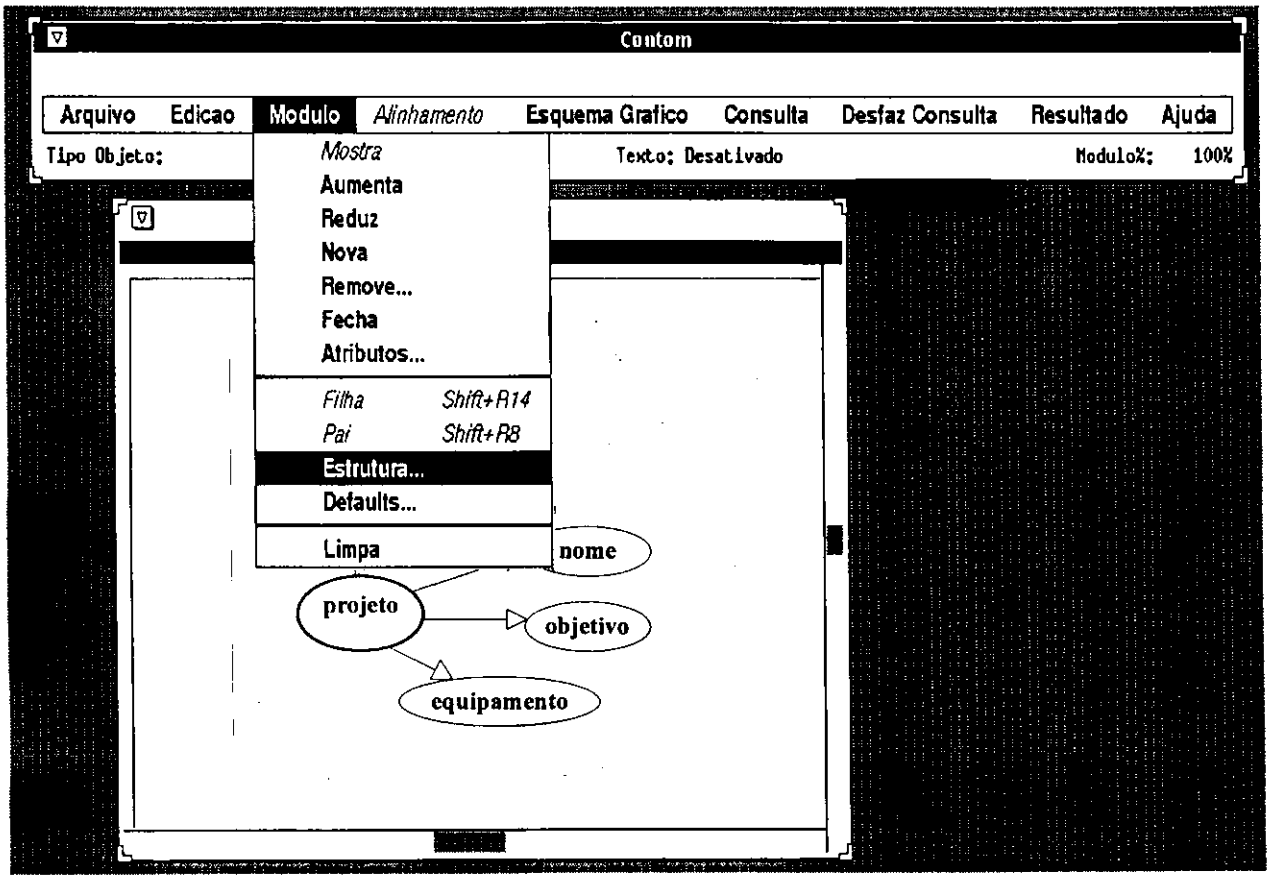


Figura 6.12. Menu *Módulo* Ativo

Capítulo 7

Conclusão

7.1. Considerações Finais

Este trabalho apresentou o ambiente gráfico de consultas a BD's - ConTOM, considerando como aspectos relevantes: a especificação visual do esquema conceitual, a apresentação dos operadores gráficos que compõem o ambiente gráfico, com a formalização dos mesmos; a realização de um estudo comparativo entre ConTOM e outras interfaces gráficas e o projeto de interface da ferramenta ConTOM. Existem duas contribuições principais neste trabalho, na área de BD's. A primeira contribuição é que o ambiente explora integralmente a representação visual, reduzindo a um mínimo necessário os dados textuais. Outra importante contribuição, é a integração desta representação visual à especificação de consultas temporais, esta característica não foi encontrada em outros ambientes gráficos.

ConTOM se fundamenta no modelo TOM, porém, os operadores gráficos são compostos de primitivas que foram formalizadas de forma independente do modelo de dados, pois se baseia em um formalismo genérico para interfaces gráficas. Assim, sua área de aplicação se estende a consultas a BD's heterogêneos - BD's caracterizados pela presença de diferentes modelos e diferentes linguagens de consulta. O formalismo, por sua vez, se baseia em conceitos de classe, relacionamento, propriedade e hierarquia *é-um*, e foi estendido para suportar consultas temporais.

Apesar de não ser necessário um conhecimento prévio do sistema para o seu uso, uma familiaridade com esquemas conceituais e consultas gráficas irá facilitar a elaboração correta de uma consulta. O usuário, portanto, pode formular consultas sem precisar conhecer aspectos relacionados ao cálculo do predicado, álgebra relacional ou a sintaxe formal de uma linguagem, aprendendo a interagir com o ambiente por tentativas, sem rigor sintático.

Outro importante aspecto encontrado em ConTOM está na resolução do problema de complexidade dos esquemas conceituais, apresentando estes esquemas dentro de uma perspectiva modular, possibilitando uma divisão prévia do ECG em módulos contextuais, visto que esquemas realísticos contém um número considerável de classes, e cada classe contém um número. Em suma, o trabalho visa transportar a simplicidade de interfaces gráficas para um ambiente de objetos complexos em que todos os dados são representados como classes de objetos, inclusive o resultado da consulta.

7.2. Sugestões de Trabalhos Futuros

Apresentamos agora, algumas idéias que surgiram no decorrer do desenvolvimento deste trabalho e que visam aprimorar o seu estado atual. Estas idéias surgiram durante a fase final do trabalho, tornando, desta forma, totalmente inviável a sua realização, podendo levar a um desvio do escopo previamente definido. Eis algumas sugestões:

- Especificação explicíta de quantificadores: Em ConTOM, não há uma especificação explícita para quantificadores na especificação de uma consulta, o quantificador implícito é o existencial - $\exists$. Portanto, não será permitido especificar consultas que exigem o uso do quantificador universal - $\forall$. Por exemplo, uma consulta tipo "*Recupere os engenheiros que trabalham em projetos de Campina Grande*", irá recuperar todos os engenheiros que trabalham em pelo menos um projeto de Campina Grande, sendo possível esta consulta ser especificada em ConTOM. Contudo, se a consulta for modificada para encontrar os engenheiros que só trabalham em projetos de Campina Grande, em ConTOM, deve ser utilizado o mecanismo de diferença como solução.

- Especificação visual e realização de consultas recursivas.

- Correspondência entre à aplicação de cada operador gráfico e a construção sintática de uma linguagem de consulta textual.

- Especificação de consultas a nível de instância: em ConTOM, a manipulação do ECG durante uma consulta é feita somente em classes e relacionamentos, os objetos só podem ser referenciados indiretamente, através da criação de uma classe contendo uma única instância. Adicionalmente, também poderiam ser permitidas consultas a nível de módulos, em situações que considerem todo o estado de um módulo ou do BD.

- Representação da negação no processo de Seleção: Como o processo de seleção já delimita o que é relevante para a consulta, poderia ser interessante considerar, no processo inicial, instâncias de uma classe que não estão associadas a instâncias de outras classes. Portanto, consultas tipo "*Recupere os empregados que não tem filhos*", poderiam ser feitas de uma forma mais intuitiva, do que a forma

empregada atualmente em ConTOM, como ilustra a figura abaixo (em ConTOM, a proposta atual para especificar a consulta é aplicar o operador de diferença):

Solução Atual **Solução Proposta**

Figura 7.1. Duas Alternativas de Negação

- Acesso à cardinalidade do relacionamento: poderia ser incluído um operador #, que pudesse fornecer a cardinalidade de um relacionamento, de forma a especificar consultas tipo "*Recupere os empregados que tem mais de três filhos*".

As sugestões acima se referem especificamente a extensões de ConTOM dentro do âmbito de consultas. Outros aspectos incluem:

- Implementação da Ferramenta ConTOM: Baseado no projeto de interface da ferramenta ConTOM, onde foram considerados alguns princípios básicos que irão servir como ponto de partida para uma implementação.

- Incorporação do ambiente ConTOM a um sistema que forneça um suporte de projeto e consultas de aplicações de Banco de Dados (BD's) convencionais ou avançados, em um ambiente relacional ou orientado a objeto (OO), tendo como principal interesse, fornecer um ambiente gráfico que ajude o projetista no desenvolvimento da aplicação, e o usuário final, na atualização e consulta ao BD. O sistema deveria fornecer ao projetista de aplicação e ao usuário final um conjunto de ferramentas (editor gráfico, linguagem de especificação, etc) que lhe permitisse: **definição** do esquema de aplicação pelo projetista, **tradução** do esquema para diversos sistemas de BD's desejados (Sistemas relacionais, Sistemas OO), **consulta** do esquema pelo usuário final (inclusão do ambiente ConTOM extendido) e **manipulação** dos dados pelo esquema gráfico.

- Completeza em relação a uma álgebra de objetos, análoga a álgebra relacional, para garantir que qualquer tipo de consulta seja realizável pelo sistema.

Referências Bibliográficas

[Abiteboul84] S. Abiteboul & R. Hull, "IFO: A Formal Semantic Database Model, em *Proceedings ACM SIGACT-SIGMOD Symp. on Principles of Database Systems,* Abril 1984, p. 119-132.

[Abiteboul91] S. Abiteboul & A. Bonner, "Objects and Views", em *ACM Transactions On Database Systems,* 1991, p. 238-247.

[Ahn86] I. Ahn, "Towards and Implementations of Database Management Systems with Temporal Support", em *Proceedings IEEE Conference on Data Engineering,* Los Angeles, 1986.

[Alashqur89] A. M. Alashqur, S. Y. W. Su & H. Lam, "OQL: A Query Language for Manipulating Object-Oriented Databases", em *Proceedings of the 15th International Conference on Very Large Data Bases,* Amsterdam, Agosto 1989, p. 433-441.

[Allen84] J. Allen, "Towards a General Theory of Action and Time", em *Artificial Intelligence, Vol. 23,* 1984, p. 123-154.

[Angelaccio90]] M. Angelaccio, T. Catarci & G. Santucci, "QBD*: A Graphical Query Language with Recursion", em *IEEE Transactions on Software Engineering, Vol. 16, Nº 10,* Outubro 1990, p. 1150-1163.

[Ariav86] G. Ariav, "A Temporally Oriented Data Model", em *ACM Transactions on Database Systems", Vol. 11, Nº 4,* Dezembro 1986, p. 499-527.

[Bancilhon89] F. Bancilhon, M. Atkinson, D. DeWitt, K. Dittrich, D. Maier & S. Zdonik, "The Object-Oriented Database System Manifesto, em *Proceedings of the First International Conference on Deductive and Object-Oriented Databases,* Kyoto, Japão, Dezembro 1989, p. 40-57.

[Brayner94] A. R. A. Brayner & C. B. Medeiros, "Incorporação do Tempo em um SGBD Orientado a Objetos, em *Anais do IX Simpósio Brasileiro de Banco de Dados,* São Carlos, Setembro 1994, p. 16-29.

[Bertino92] E. Bertino, M. Negri, G. Pelagatti & L. Sbatella,"Object-Oriented Query Languages: The Notion and Issues", em *IEEE Transactions on Knowledge and Data Engineering, Vol. 4, No 3,* Junho 1992, p. 223-237.

[Bratko86] I. Bratko, "Prolog - Programming for Artificial Intelligence", *UK: Addison Wesley,* 1986.

[Bryce86] D. Bryce & R. Hull, "SNAP: A Graphics-based Schema Manager", em *Proceedings of the 2nd International Conference on Data Engineering,* Los Angeles, 1986, p. 151-164.

[Carvalho93] A. Carvalho, "Tom-Rules: Um Monitor de Eventos, Regras e Gatilhos em um Ambiente Orientado a Objetos", *Dissertação de Mestrado,* UFPB, Campina Grande, 1993.

[Catarci93] T. Catarci, G. Santucci & M. Angelaccio, "Fundamental Graphical Primitives for Visual Query Languages", em *Informations Systems, Vol. 18, No 2,* 1993, p. 75-98.

[Cavalcanti94] A. E. C. Cavalcanti, "Um Estudo para tratar a Dimensão de Tempo em Sistemas de BD's", em *Anais do IX Simpósio Brasileiro de banco de Dados,* São Carlos, Setembro 1994, p. 357-381.

[Chamberlin74] D. D. Chamberlin & R. F. Boyce, "Sequel: A Structured English Query Language", em *Proceedings ACM-SIGDIFET Workshop,* 1974.

[Chen76] P. P. S. Chen, "The Entity-Relationship Model - Toward a Unified View of Data, em *ACM Transactions on Database Systems, Vol. 1, No 1,* março 1976, p. 9-36.

[Clifford85] J. Clifford,"Towards an Algebra of Historical Relational Databases", em *ACM SIGMOD Conference,* 1985.

[Clifford87] J. Clifford & A. Crocker, "The Historical Relational Data Model (HDRM) and Algebra Based on Lifespans", em *Proceedings International Conference on Data Engineering, 3,* Fevereiro 1987, Los Angeles, California, p. 528-537.

[Codd70] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks", *em ACM Communications, Vol. 13, No 6,* Junho 1970, p. 377-387.

[David92] M. B. David, "Descrição Formal da Estrutura do Modelo Orientado a Objetos Temporal - TOM (Temporal Object Model)", *Dissertação de Mestrado*, UFPB, Campina Grande, 1992.

[Dubois86] E. Dubois et al., "A Knowledge Representation Language for Requirements Engineering", em *Proceedings of the IEEE, Vol. 74*, No 10, Outubro 1986, p. 1431-1444.

[Edelweiss94] N. Edelweiss & J. P. M. de Oliveira, "Modelagem de Aspectos Temporais de Sistemas de Informação", em *IX Escola de Computação*, Recife, Julho 1994.

[Elmasri90] R. Elmasri & G. T. J. Wuu, "A Temporal Model and Query Language for ER Databases", em *Proceedings of International Conference on Data Engineering, 6.*, Los Angeles, Fevereiro 1990, p. 76-83.

[Foley82] J. D. Foley & A. Van Dam, "Fundamentals of Interactive Computer Graphics". *Addison-Wesley*, Reading, Mass.

[Furtado93a] M. E. S. Furtado, "Uma Metodologia para Projeto de Banco de Dados Temporal Orientado a Objetos", em *Anais do VIII Simpósio Brasileiro de banco de Dados,* Campina Grande, Maio 1993, p. 314-327.

[Furtado93b] M. E. S. Furtado, "Uma Metodologia para Projeto de Banco de Dados Temporal Orientado a Objetos", *Dissertação de Mestrado*, UFPB, Campina Grande, 1993.

[Gadia88] S. K. Gadia, "A Homogeneous Relational and Query Language for Temporal Databases" *em ACM Transaction on Database Systems, Vol. 13, No 4*, 1988.

[Hull87] R. Hull & R. King, "Semantic Database Modeling: Survey, Applications and Research Issues" em *ACM Computing Surveys, Vol. 19, No 3*, Setembro 1987, p. 201-260.

[Hull89] R. Hull "Four Views of Complex Objects: A Sophisticated Introduction" em *Lecture Notes in Computer Science, Vol. 361*, S. Abiteboul, PC Fischer, H. Sheck (Eds).

[Jensen94] C. S. Jensen & al., "A Consensus Glossary of Temporal Database Concepts", em *SIGMOD-RECORD, Vol. 23, No 1*, Março 1994.

[Kãfer90]] W. Kãfer, N. Ritting & H. Schõning, "Support for Temporal Data by Complex Objects", em *Proceedings of 16th Very Large Data Bases,* Brisbane, Australia, 1990, p. 24-35.

[Kent79] W. Kent, "Limitations of Record-Based Informations Models", em *ACM Transactions on Database Systems, Vol. 4, No 1*, Janeiro 1979, p. 107-131.

[KimHJ88] H. J. Kim, H. F. Korth & A. Silberchatz, "PICASSO: A Graphical Query Language", em *Software-Practice and Experience, Vol. 18, No 3*, Março 1988, p. 169-203.

[Kim89] W. Kim, "A Model of Queries for Object-Oriented Data Bases", em *Proceedings of the 15th International Conference on Very Large Data Bases,* Amsterdam, Agosto 1989, p. 423-432.

[Khoshafian90] S. Khoshafian & R. Abnous, "Object Orientation: Concepts, Languages, Databases, User Interfaces", *John Wiley & Sons, Inc.*, 1990.

[Kuntz90] M. Kuntz, "Description et Évaluation de PASTA-3, une interface graphique à manipulation directe aux bases de donées avancées", em *Vlémes Journées Bases de Données Avancées,* Montpellier, Setembro 1990, p. 123-140.

[Lorentzos88] N. A. Lorentzos & R. Johnson, "Extending Relational Algebra to Manipulate Temporal Data", em *Information Systems, Vol. 13, No 3*, 1988, p. 289-296.

[Lorentzos93] N. A. Lorentzos, "The Interval-extended Relational Model and its Applications to Valid-time Databases", em *Temporal Databases, A. U. Tanzel et al.* (eds.), Redwood, California: Benjamin/Cummings, 1993, p. 67-91.

[Loucopoulos91] P. Loucopoulos, B. Theodoulidis & D. Pantazis, "Business Rules Modelling: Conceptual Modelling and Object-oriented Specifications", em *Proceedings of the*

IFIP TC8/WG8.1 Working Conference, Quebec, Canada, Outubro 1991, p. 323-342.

[Mark87] L. Mark, "A Graphical Query Language for the Binary relationship Model", em *Information Systems, Vol. 14, No 3*, 1989, p. 231-246.

[McDonald75] N. McDonald & M. Stonebraker, "CUPID: A User Friendly Graphics Query Language", em *Proceedings ACM-PACIFIC,* Abril 1975, p. 127-131.

[MetaSoft89] Metasoftware Corporation, *Design/OA - Developer's Manual*, 1989.

[MetaSoft93] Metasoftware Corporation, *MetaDesign - User's Guide*, 1993.

[Mohan90] L. Mohan, "A Framework for Building Knowledge-Intensive Data Models", em *Proceedings 2th International Conference on Software Engineering & Knowledge Engineering",* Junho 1990, p. 33-38.

[Mohan93] L. Mohan & R. L. Kashyap, "A Visual Query Language for Graphical Interaction with Schema-Intensive Databases", em *IEEE Transactions on Knowledge and Data Engineering, Vol. 5*, No 5, Outubro 1993, p. 843-857.

[Naur63] P. Naur, "Report on the Algorithmic Language ALGOL 60", em *ACM., Vol. 6*, 1963, p. 1-17.

[Navathe87] S. B. Navathe & R. Ahmed, "TSQL - A Language for History Databases", em *Proceedings TAIS Conference,* Sophia-Antipolis, 1987.

[Navathe93] S. B. Navathe & R. Ahmed, "Temporal extensions to the Relational Model and SQL", em *Temporal Databases,* A. U. Tanzel et al. (eds.), Redwood, California: Benjamin/Cummings, 1993, p. 92-109.

[Oliveira93] J. L. de Oliveira & R. O. Anido, "Navegação e Consulta em Banco de Dados Orientados a Objetos", em *Anais do VIII Simpósio Brasileiro de banco de Dados,* Campina Grande, Maio 1993, p. 35-49.

[Paredaens92] Jan Paredaens & al, "An Overview of GOOD", em *SIGMOD RECORD, Vol. 21, No 1*, Março 1992, p. 25-31.

[Poncelet93] P. Poncelet, "Contribution à La Conception des bases de Donées Avancées: Modelisation, Evolution et Derivation", *Tese de Doctorat,* Université de Nice, Nice, 1993.

[Ramos92] H. B. Ramos, "IQl: An Integrated Graphical User Interface for Relational Queries with Genericity", em *Anais do VII Simpósio Brasileiro de banco de Dados,* Porto Alegre, Maio 1992, p. 431-446.

[Rumbaugh87] J. Rumbaugh, "Relations as semantic Constructs in an Object-Oriented Language", em *Proceedings OOPSLA 87*, Outubro 87, p. 466-481.

[Rundensteiner92] E. A. Rundensteiner & L. Bic, "Set Operations em Object-Based Data Models", em *IEEE Transactions on Knowledge and Data Engineering, Vol. 4, No 3*, Junho 1992, p. 223-237.

[Sarda90] N. L. Sarda, "Extensions to SQL for Historical Databases", em *IEEE Transaction on Knowledge and Data Enginnering, Vol. 2, No 2*, Junho 1990, p. 220-230.

[Schek90] H. J. Schek & M. H. Sholl, "Evolution of Data Models", *Lectures Notes in Computer Science,* Vol. 466, Blaser A (ed) "Database Systems for 90's", Springer-Verlag, 199, p. 135-153.

[Schiel83] U. Schiel, "An Abstract Introduction to the Temporal-Hierarchic Data Model (THM)", em *Proceedings of the 9th International Conference on Very Large Data Bases,* Florence, 1983, p. 322-330.

[Schiel84] U. Schiel, A. L. Furtado, E. J. Neuhold & M. A. Casanova, "Towards Multi-Level and Modular Conceptual Schema Specifications", em *Information Systems, Vol. 9, No 1*, Junho 1984, p. 43-57.

[Schiel89] U. Schiel, "Vodak Version Model (VVM)", *Working Paper,* GMD/IPSI, Darmstadt/Germany, 1989.

[Schiel90] U. Schiel & I. Mistrik, "Using Object-Oriented Analysis and Design for Integrated Systems", em *Proceedings of International Conference on Systems Integration, IESI,* New Jersey 1990.

[Schiel91] U. Schiel, "An Open Environment for Objects with Time and Versioning", *em Proceedings EastEurOOpe*, Bratislava, 1991, p. 116-125.

[Schiel93] U. Schiel, "Aspectos Temporais em Sistemas de Informação", *Tese (concurso público para prof. titular)*, UFPB, Campina Grande, Janeiro 1993.

[Schneider91] M. Schneider & C. Trepied, "Puissance D'expression du Langage Graphique D'interrogation CANDID", em *VIIémes Journées Bases de Donnés Avancées*, Lyon, Setembro 1991, p. 409-426.

[Segev87] A. Segev & A. Shoshani, "A Logical Modeling of Temporal Data", em *Proceedings of ACM SIGMOD International Conference on Management of Data*, San Francisco California, Maio 1987, p. 454-466.

[Shneiderman83] B. Shneiderman, "Direct Manipulation, a Step Beyond Programming Languages", em *IEEE computer, Vol. 16, No 8*, Agosto 1983, p. 57-69.

[Shneiderman87] B. Shneiderman, "Designing The User Interface". *Addison-Wesley*, 1987.

[Snodgrass87] R. Snodgrass, "The Temporal Query Language TQUEL", *em ACM-Transaction on Database Systems, Vol. 12, No 2*, 1987.

[Sockut93] G. H. Sockut, L. M. Burns, A. Malhotra & Kyu-Young, "GRAQULA: A Graphical Query Language for Entity-Relationship or Relational Databases" *em Data & Knowledge Engineering, 11*, 1993, p. 171-202.

[Stanley91] Stanley Y. W. Su, Hsin-Hsing & M. Chen, "A Temporal Knowledge Representation Model OSAM*/T and Its Query Language OQL/T", em *Proceedings of the 17th International Conference on Very Large DataBases*, Barcelona, Setembro 1991, p. 431-442.

[Stonebraker76] M. R. Stonebraker, E. Wong, P. Kreps & G. Held, "The Design and Implementation of INGRES", em *ACM Transaction Database Systems, Vol. 1, No 3*, Setembro 1976, p. 189-222.

[Sun90] Sun Microsystems Inc., *Open Windows Developer's Guide - Users Manual*, 1990.

[Tanzel85] A. U. Tanzel, "Adding Time Dimension to Relational Model and Extending relational Algebra", em *Information Systems, Vol. 11, No 4*, 1986, p. 343-355.

[Teorey86] T. J. Teorey, D. Yang & J. Fry, "A Logical Design Methodology For Relational Databases using the Extended Entity Relationship Model", *em Computing Surveys, Vol. 18, No 2*, Junho 1986, p. 197-222.

[Theodoulidis91] C. Theodoulidis, P. Loucopoulos & B. Wangler, "A Conceptual Modeling Formalism for Temporal Database Applications", em *Information Systems, Vol. 16, No 4*, 1991, p. 401-416.

[Ty88] F. Ty, "G-OQL: Graphics Interface to the Object-Oriented Query Language OQL", *Master Thesis*, University of Florida, 1988.

[Vadaparty93] K. Vadaparty, Y. A. Aslandogan & G. Ozsoyoglu, "Towards a Unified Visual Database Acess", em *ACM-SIGMOD,* Washington, DC, USA, Maio 1993, p. 357-366.

[Weiland93] W. J. Weiland & B. Shneiderman, "A Graphical Query Interface Based on Aggregation, Generalization Hierarchies", em *Information Systems, Vol. 18, No. 4*, 1993, p. 215-232.

[Wong82] H. K. T. Wong & I. Kuo, "GUIDE: A graphical user interface for database exploration", em *Proceedings 8th International Conference on Very Large Databases,* 1982, p. 22-32.

[Wuu93] G. T. J. Wuu & U. Dayal, "A Uniform Model for Temporal and Versioned Object-oriented Databases", em *Temporal Databases: Theory, Design and Implementation.* Bridge Parkway: Benjamin/Cummings, 1993, p. 230-247.

[Zloof77] M. M. Zloof, "Query-By-Example: A Database Language" em *IBM System Journal, Vol. 16, No 4*, 1977, p. 324-343.