Universidade Federal da Paraíba
Centro de Ciências Tecnológicas
Departamento de Sistemas e Computação

Uma Metodologia para Projeto de Banco de Dados Temporal Orientado a Objetos

Dissertação de Mestrado

por

Maria Elizabeth Sucupira Furtado

Orientador: Ulrich Schiel

Campina Grande, Agosto 1993

Uma Metodologia para Projeto de Banco de Dados Temporais Orientado a Objetos

Este trabalho foi apresentado à Pós-Graduação em Informática do Centro de Ciências e Tecnologia da Universidade Federal da Paraíba como requisito parcial para obtenção do grau de Mestre em Informática.

Orientador: Ulrich Schiel

Campina Grande, Agosto de 1993

F992m Furtado, Maria Elizabeth Sucupira
Uma metodologia para projeto de banco de dados temporal orientado a objetos / Maria Elizabeth Sucupira Furtado. - Campina Grande, 1993.
137 f. : il.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Sistemas Orientados a Objetos 2. Banco de Dados Temporais e Ativos Orientados a Objetos 3. Dissertacao I. Schiel, Ulrich, Dr. II. Universidade Federal da Paraiba - Campina Grande (PB) III. Título

CDU 004.652.5(043)

UMA METODOLOGIA PARA PROJETO DE BANCO DE DADOS TEMPORAL ORIENTADA A OBJETOS.

MARIA ELIZABETH SUCUPIRA FURTADO

DISSERTAÇÃO APROVADA EM 09.08.1993

ULRICH SCHIEL Dr.
ORIENTADOR

MARIA DE FÁTIMA Q.V. TURNELL Ph.D.
Componente da Banca

DÉCIO FONSECA Dr.
Componente da Banca

Campina Grande, 09 de agosto de 1993

Agradecimentos

Ao meu marido Vasco, pelo amor, amizade e críticas inteligentes, essenciais para o aperfeiçoamento desse trabalho.

Aos meus pais e irmãos, pelo constante carinho e pelo desejo de meu sucesso.

À D.Lúcia, minha querida sogra, pela participação sempre que necessária à nossa adaptação nessa cidade.

Ao professor e amigo Ulrich Schiel, pela competente orientação fornecida e pela constante disponibilidade para esclarecer dúvidas surgidas durante o trabalho.

À professora Fátima Turnell, sempre presente nos momentos em que necessitei, fornecendo sugestões e ajudando no meu aperfeiçoamento profissional.

Ao professor Décio Fonseca, pela participação na banca examinadora e pelos comentários feitos sobre o trabalho.

À amiga Emilia, pelo apoio na pesquisa bibliográfica e por resolver meus assuntos profissionais.

Ao Núcleo de processamento de dados da UFCE pela oportunidade que me foi dada.

À CAPES pelo apoio financeiro.

RESUMO

O projeto de sistemas orientado a objetos é uma nova visão de projeto que enfatiza a definição de características estruturais e comportamentais dentro de um sistema de objetos. Ele tem trazido as seguintes vantagens: melhor poder de expressividade para modelagem das aplicações, melhor estrutura baseada no conceito de tipos abstratos de dados e objetos complexos, possibilidade de reusar um software durante o desenvolvimento do sistema e continuidade conceitual durante todas as fases do ciclo de vida.

Devido a estes motivos nos deparamos com uma proliferação de metodologias de desenvolvimento orientadas a objetos, ficando difícil ao projetista escolher aquela que melhor se adapte a uma determinada aplicação. Sugerimos a aplicação de alguns critérios de avaliação de qualidade para atenuar este problema.

Avaliamos em detalhe as dificuldades, restrições e peculiaridades de algumas metodologias de desenvolvimento orientadas a objetos. A partir deste estudo comparativo apresentamos uma nova metodologia (FADO) capaz de cobrir alguns pontos encontrados como deficientes. Todas as metodologias são descritas com relação a sua representação gráfica e textual e ao processo.

A metodologia proposta auxilia no projeto de banco de dados temporais e ativos orientados a objetos modelando o domínio em objetos e eventos. O processo FADO permite a reusabilidade, iteração e uso de protótipos como forma de melhor cobrir à extensibilidade e alterabilidade.

A metodologia fundamenta-se nas metodologias de Booch, Rolland e Schiel, visando a geração da especificação de um esquema conceitual no Modelo Temporal de Objetos (TOM).

Neste trabalho, também é descrito a interface da ferramenta automatizada FADO, que dá suporte à utilização das técnicas que compõem a metodologia.

ABSTRACT

The object-oriented design is a new view of design which defines structures and behavior of objects systems. This design presents some advantages: better expressiveness power for application modelling, better structure based on the concepts of abstract data types and complex objects, reusability during the systems development process and conceptual continuity during all life cycle phases.

This fact has contributed for growing of several OO methodologies. Therefore it becomes difficult for the OO designer to choose the most suitable methodology for a specific application. We suggest some criteria to evaluate and compare current object-oriented analysis and design techniques.

In this work we find the difficulties, restrictions and characteristics of some OO methodologies and propose a new OO methodology (FADO) covering some of these problems. All the methodologies are described considering their representation and process.

The FADO methodology supports active temporal object oriented database design, modelling the domain into objects and events. The FADO process permites reusability, iteraction and use of prototypes as the best way of covering the extensibility and maintenability.

The FADO methodology is based on the approaches of Booch, Rolland and Schiel and generates a conceptual schema in the Temporal Object Model (TOM).

In this work, the FADO tool is also described which gives a computational support to the methodology.

Sumário

Capítulo 1 - Introdução

Capítulo 2 - Evolução da abstração na modelagem conceitual - Em direção a orientação a objetos

Capítulo 3 - Banco de dados orientados a objetos

Capítulo 4 - Avaliação das metodologias de desenvolvimento orientadas a objeto

Capítulo 5 - Metodologia FADO

Capítulo 6 - Estudo de caso

Capítulo 7 - Ferramenta FADO

Capítulo 8 - Conclusão

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

1.1 A abrangência de orientação a objetos

Enquanto computadores mais sofisticados estão sendo fabricados, a inabilidade em apresentar técnicas e conceitos de software mais flexíveis e fáceis de usar, que dêem suporte aos sistemas que estão cada vez mais complexos, constitui um crescente problema. Existe um estudo indicando que até o início dos anos 90, o software estava pelo menos duas gerações de processadores atrás do hardware [Winblad93]. A filosofia de orientação a objetos surge como solução promissora para resolver esses problemas. A orientação a objetos permite aos projetistas controlar a complexidade do software fazendo uso dos novos conceitos para construção de sistemas, como: objetos, classes, métodos, mensagem e hereditariedade. A utilização desses conceitos vem produzindo benefícios em termos de tempo de desenvolvimento, recursos de programação e habilidade na construção de novas gerações de software.

A influência da orientação a objetos se reflete nas duas principais áreas que dão apoio à construção de sistemas: banco de dados e engenharia de software (linguagem de programação, ambiente de desenvolvimento de sistemas e metodologia de desenvolvimento).

Os bancos de dados orientados a objetos (BDOO) são capazes de dar suporte às aplicações complexas de forma mais efetiva do que os outros modelos, além de solucionar o problema da compatibilidade das linguagens de definição e de manipulação dos dados. Os BDOO oferecem as mesmas fucionalidades existentes nas linguagens de programação orientada a objetos, como encapsulamento, hereditariedade, amarração dinâmica, polimorfismo e apresentam características que lhes são peculiares: persistência, integridade, concorrência, segurança e linguagens de consulta.

A programação orientada a objetos fornece oportunidade de desenvolver aplicações baseadas no trabalho de terceiros (as classes podem ser misturadas e modificadas para criação de novas aplicações) e de suportar estrutura de dados mais complexas. O principal objetivo dos que optam por uma linguagem orientada a objetos é de desenvolver programas mais rapidamente e fáceis de modificar. Nesta nova filosofia, o programador não se resume a escrever programas para resolver o problema, mas em encontrar classes já existentes que facilitem a sua solução.

Os ambientes de desenvolvimento de software contam com interfaces que encorajam os usuários a pensar em todos os elementos visuais (ícones) como objetos a serem manipulados diretamente com o mouse e o teclado, tendo o suporte de um sistema orientado a objetos.

Quanto às metodologias, podemos dizer que as que se baseiam em orientação a objetos possuem melhor poder de expressividade para modelagem das aplicações, melhor estrutura baseada no conceito de tipos abstratos de dados, possibilidade de reusar software durante o desenvolvimento do sistema e continuidade conceitual durante todas as fases do ciclo de vida.

Nos anos 90, as aplicações orientadas a objetos alcançarão, provavelmente, presença e popularidade. O sucesso comercial da orientação a objetos dependerá da integração dos sistemas, linguagens, ferramentas, banco de dados e biblioteca de classes para dar suporte à criação e implementação de sistemas de grande escala e, principalmente, de metodologias sofisticadas apropriadas para reduzir a complexidade do problema, melhorar a comunicação usuário-analista e modelar dados complexos.

Uma metodologia deve ser utilizada para apoiar as atividades fundamentais de desenvolvimento de um sistema de computador. Ela ajuda o projetista a organizar o pensamento do propósito do problema para descobrir o que precisa ser definido, quais as alternativas existentes para solução do problema, qual a melhor escolha e a fazer a consistência de suas definições.

Neste trabalho, analisamos em detalhe as dificuldades, restrições e peculiaridades de algumas metodologias de desenvolvimento orientadas a objetos e propomos uma nova metodologia, chamada FADO (Ferramenta de Análise e Desenvolvimento Orientada a Objetos)[Furtado93]. Como durante todo este tabalho nos referiremos aos conceitos de orientação a objetos, a próxima seção oferece definições básicas sobre esses conceitos.

1.2. Conceitos fundamentais de orientação a objetos

O paradigma de orientação a objetos deve ser apresentado como um modelo baseado em objetos que interagem entre si, através de troca de mensagens, para realizar uma determinada tarefa.

Objeto é qualquer entidade do mundo real, que consiste de uma estrutura definida por um conjunto de atributos e operações que atuam sobre ele. As operações contidas dentro do objeto levam o nome de método. Em vez de dados passivos de um programa convencional, os objetos podem atuar e ser ativados por meio de mensagens, a partir de outros objetos.

A seguir classificamos três conceitos fundamentais de orientação a objetos : identidade de objetos, tipos abstratos de dados e herança.

1.2.1. Identidade de objeto

Identidade de um objeto é a propriedade que distingue um objeto de outro. Fazendo uma analogia ao conceito de variável nas linguagens de programação, podemos dizer que mesmo que duas variáveis tenham mesmo valor elas têm sua identidade própria (sua posição em memória). Com relação aos objetos o mesmo acontece, cada objeto possui sua identidade mesmo que possuam mesmo valores.

Objetos são considerados transitórios, porque podem ser criados, modificados e destruídos durante a execução de um programa. Suas características dinâmicas estão associadas a uma série de conceitos:

- Objetos têm um estado, que se refere a uma possível situação vivenciada por ele, ao adquirir valores específicos nos seus atributos.
- Objetos exibem um comportamento que define como ele age ou reage diante de uma mudança de estado e troca de mensagens. Por exemplo, suponha que um processo mande a mensagem *desempilhe* para o objeto *fila*, a reação deste é encolher a fila.
- Objetos podem ser classificados em função de suas semelhanças em classes de objetos. Se as instâncias de classes identificadas apresentam atributos comuns, então podemos agrupá-las em superclasses, e assim sucessivamente.
- Objetos podem mudar de classe em função de mudanças no seu estado que podem requerer uma nova modificação. Por exemplo, quando uma pessoa *Desempregada* obtiver um emprego, ela passará a fazer parte da classe *Empregado*.
- Objetos podem ser criados ou destruídos
  Um objeto começará a existir a partir da instanciação em uma classe e permanecerá vivo até ser explicitamente eliminado (através de uma mensagem) ou, alternativamente, até o instante em que não haja mais nenhuma referência a ele [Takahashi90].
- Objetos podem ser complexos representando uma estrutura de composição aninhada de outros objetos.

1.2.2. Tipos abstratos de dados

Tipos abstratos de dados combinam a noção de tipos de dados, com o ocultamento de informação, isto é, todas as variáveis de um tipo de dados só podem ser manipuladas, através de operações pré-definidas associadas ao tipo.

Uma importante extensão deste conceito para conseguir um maior poder da modelagem semântica, é o uso das formas de abstração: generalização, agregação e agrupamento.

Podemos definir classe como um mecanismo de abstração de dados, que permite fatorar propriedades

estáticas (atributos) e dinâmicas (métodos) comuns a objetos, isto é, objetos com a mesma estrutura e comportamento são considerados instâncias de uma mesma classe. Desta forma é evitada uma descrição individual de cada objeto em uma aplicação.

Esse mecanismo de abstração é composto de duas partes: uma parte da interface e uma parte da implementação. Considerando esta divisão, a única parte visível para o usuário da classe são os nomes e parâmetros das operações da interface. Qualquer objeto que seja instância daquela classe só poderá ser manipulado por meio dos métodos definidos na interface. Tal característica é denominada encapsulamento, cujo objetivo é fornecer proteção aos objetos, assegurando a consistência e a integridade de suas estruturas.

Se classes e tipos abstratos de dados descrevem uma coleção de objetos com a mesma estrutura e comportamento, qual a diferença entre eles?

- Tipos abstratos de dados são implementados através de classes, e as classes incorporam a definição da estrutura e das operações de um tipo abstrato de dado [Meyer88];
- A noção de tipo permite a checagem de tipo em tempo de compilação, enquanto que a noção de classe permite a checagem de tipo em tempo de execução, para suportar o conceito de polimorfismo.
- As linguagens que suportam tipos abstratos de dados só permitem acesso público à interface, enquanto algumas linguagens que suportam classes, permitem acesso público, privado e protegido.

1.2.3. Herança

É um mecanismo de reusabilidade de atributos e métodos de objetos que fazem parte de uma estrutura hierárquica. Ao criarmos uma classe subordinada a outra, denominada superclasse, a nova classe, denominada subclasse, herda todas suas propriedades: atributos e métodos.

O acoplamento do método à mensagem, na hierarquia de classes, é feito em tempo de execução e obedece ao seguinte critério: o método a ser executado é procurado na classe cujo objeto é instância. Se não for encontrado, então será procurado na superclasse dessa classe e assim sucessivamente. A pesquisa pára ao encontrá-lo, então o acoplamento é realizado e o método é executado. Este processo é denominado amarração dinâmica.

1.3. Objetivo da dissertação

Existem dois objetivos fundamentais neste trabalho. O primeiro é fazer uma análise comparativa entre

as metodologias de Shlaer, Schiel, Rolland, Booch e Baptista, seguindo alguns critérios de avalição de qualidade. O segundo objetivo é propor uma nova metodologia de análise e projeto de bancos de dados temporais orientados a objetos para o ambiente DYNAMO [Schiel92].

A motivação deste trabalho foi devido aos problemas encontrados com as metodologias orientadas a objetos existentes, como: modelagem de características temporais, identificação e representação de outros tipos de relacionamentos e abstrações, otimização de classes, representação do comportamento dinâmico (troca de mensagens associadas ao controle de eventos) e integração dos modelos estáticos e dinâmicos.

O objetivo básico da metodologia FADO é representar a estrutura e comportamento das classes, através da análise de eventos (controle, sequência e tempo) fazendo uso de um processo iterativo e incremental. A metodologia FADO tem como fundamento a flexibilidade e a iteração, onde os passos a serem seguidos não obedecem a uma sequência rígida.

O ambiente DYNAMO objetiva dar suporte ao desenvolvimento de aplicações mais avançadas de sistemas de processamento da informação e do conhecimento. A arquitetura geral desse ambiente pode ser vista na figura 1.1. Nela podemos visualisar uma interface de manipulação, onde o usuário faz seus requisitos ao sistema e recebe os resultados. Esses requisitos e resultados são processados pelo gerenciador de objetos. O gerenciador de objetos faz todas as conversões entre o ambiente de aplicação e o ambiente de gerência das informações e monitora os diversos acessos aos objetos. O monitor de eventos detecta a ocorrência de eventos pela análise de mensagens recebidas do gerenciador de objetos ou pela consulta ao relógio interno e solicita ao gerenciador de objetos a execução das ações especificadas de acordo com o esquema conceitual.

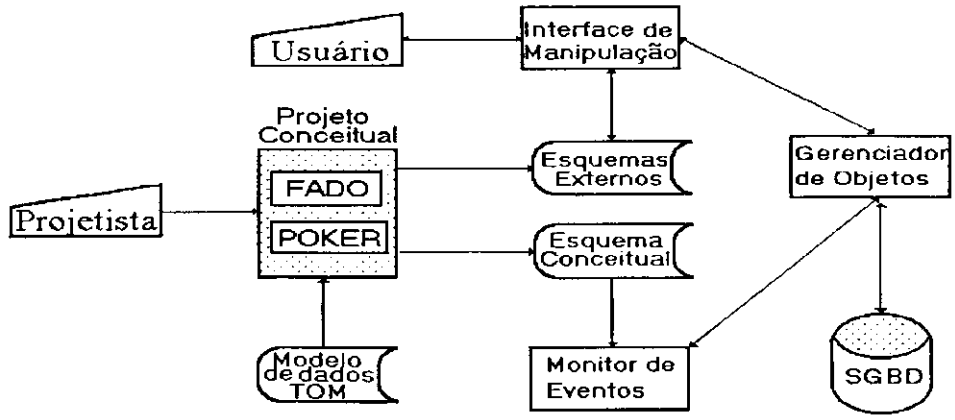

Figura 1.1 - Ambiente DYNAMO

O modelo conceitual utilizado no DYNAMO é o Modelo Temporal de Objetos (TOM) [Schiel91]. As ferramentas de projeto conceitual são POKER [Schiel90] e FADO. Ambas terão um suporte computacional para apoiar o desenvolvimento dos esquemas externos ou conceituais em TOM. Essas metodologias serão vistas no decorrer deste trabalho.

1.4. Estrutura da dissertação

Este trabalho é dividido em 4 partes. A primeira parte é composta dos capítulos 2 e 3 e se caracteriza por uma abordagem conceitual para situar o leitor. No capítulo 2, analisamos o papel da abstração nas principais áreas que dão apoio à construção de sistemas, no capítulo 3 mostramos as características dos bancos de dados orientados a objetos e o modelo de dados TOM. A segunda parte é composta somente do capítulo 4, contendo uma análise comparativa entre cinco metodologias orientadas a objetos de acordo com alguns critérios de avaliação de qualidade mencionados. A terceira parte, composta dos capítulos 5,6 e 7 e do apêndice A, se refere a nova metodologia proposta. No capítulo 5, mostramos o processo e representação da metodologia FADO. No capítulo 6, fazemos a validação da metodologia aplicando um estudo de caso real. No capítulo 7 descrevemos o projeto de uma possível interface para a ferramenta FADO e no apêndice A apresentamos o esquema de dados TOM. A última parte se refere às considerações finais do trabalho desenvolvido.

CAPÍTULO 2

Evolução da abstração na modelagem conceitual (Em direção a orientação a objetos)

2.1. Abstração na modelagem conceitual

A análise de domínio da aplicação e a consequente modelagem de suas entidades e propriedades em uma forma operacionalizável pelo computador é denominada modelagem conceitual. Para bem realizar essa modelagem é extremamente importante utilizar o conceito de abstração.

A abstração é uma descrição ou especificação de um sistema, onde são consideradas as propriedades relevantes e desprezadas as irrelevantes, com o intuito de construirmos um modelo conceitual para o domínio observado. As características encontradas por um processo de abstração variam conforme a perspectiva do observador. A abstração é uma das maneiras fundamentais que temos para controlar a complexidade do domínio observado, solucionando problemas de alto custo e baixa qualidade dos sistemas.

As abstrações podem ser definidas em um alto nível, como objetos, ou num nível de abstração mais baixo, como estruturas de dados (pilha, lista, grafo, etc). A nossa idéia neste capítulo é mostrar como as abstrações de alto nível respondem a complexidade de desenvolvimento de sistemas.

Essas abstrações auxiliam a modelagem conceitual nas principais áreas da informática que dão apoio à construção de sistema: banco de dados e engenharia de software. Cada área aborda diferentemente a abstração na modelagem conceitual de aplicações.

2.2. Abstração na área de banco de dados

Foi nesta área que a tarefa de modelagem conceitual se tornou mais relevante. Tal sucesso se deu por

podermos seguir as convenções de representação de informações bem definidas dos modelos de dados. Essas convenções são conceitos que descrevem a estrutura (tipo de dados, relacionamentos e restrições) e as operações de um banco de dados. Este suporte dos modelos de dados à modelagem conceitual objetiva diminuir a diferença semântica existente entre o domínio da aplicação e o banco de dados, representando ao máximo a realidade dentro do próprio banco de dados.

O papel fundamental da abstração na área de banco de dados se refere à independência dos dados [McLeod85]. Os princípios da independência de dados são: organizar a estrutura física do banco de dados sem mudar a estrutura lógica e liberar o projetista de preocupações com armazenamento físico e detalhes de acesso ao banco de dados. O mecanismo de abstração que dá suporte à independência de dados é denominado de modelo de dados.

A arquitetura do segundo modelo de referência de banco de dados ANSI/X3/SPARC [Mark85] é uma tentativa de generalizar os vários estudos que têm sido feitos para descrever os dados num banco de dados. Essa arquitetura descreveu os dados nas dimensões ponto-de-vista e intenção-extensão. A primeira dimensão engloba os três níveis de abstração de dados bastante conhecidos (figura 2.1(a)): **externo** (representa os dados externos), **conceitual** (representa o significado dos dados) e **interno** (representa a estrutura de armazenamento e métodos de acesso ao banco de dados).

A dimensão intenção-extensão significa que cada nível é a extensão do nível de cima e a intenção do nível de baixo. Os quatro níveis de descrição dos dados são (figura 2.1(b)): **meta-esquema dos modelos de dados** (contém informações para definir e modificar um ou mais modelos de dados), **esquema do modelo de dados** (contém informações de um modelo de dados específico), **esquema de aplicação** (contém informações de uma aplicação do mundo real) e, por último, **os dados da aplicação.**

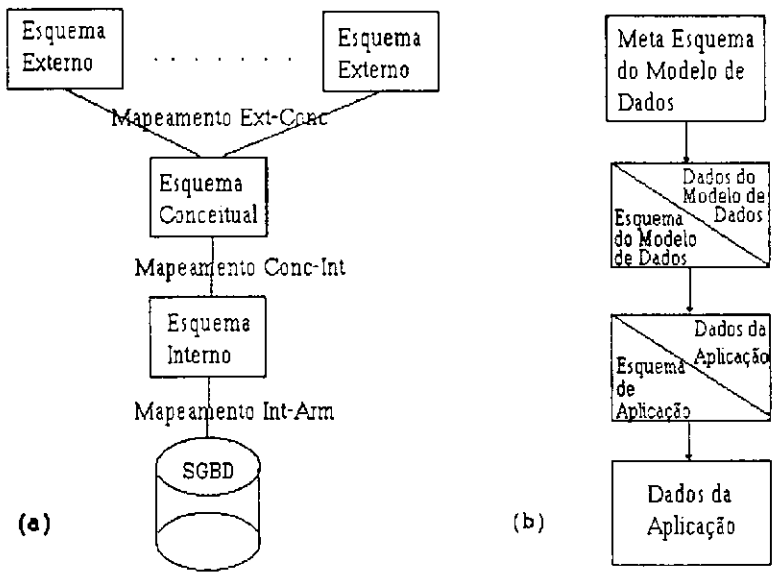

Figura 2.1 - Terminologia ANSI/SPARC

Neste trabalho focalizamos os modelos de dados em duas perspectivas distintas: na primeira perspectiva está o processo de projeto de banco de dados e na segunda estão os modelos em si, divididos em clássicos, semânticos e orientados a objetos [Navathe92].

O processo de projeto de banco de dados consiste em mapear as especificações dos dados e aplicações num banco de dados. Esse processo pode ser seguido por qualquer uma das estratégias: *bottom-up* ou *top-down*. Os passos do processo *bottom-up* são: projeto de visão (identifica como os dados são vistos por cada usuário), projeto integração (integra a percepção dos diversos usuários sobre seus dados) e projeto lógico/físico (a partir dos dados conceituais obtém-se as estruturas internas e técnicas de acesso ao banco de dados).

Os passos do processo *top-down* são: projeto conceitual (define a especificação da aplicação como um todo), projeto de visão (desenvolve os subesquemas), projeto lógico/físico (a partir dos dados conceituais obtém-se as estruturas internas e técnicas de acesso ao banco de dados).

A figura 2.2 representa o relacionamento entre a dimensão ponto-de-vista da terminologia ANSI/SPARC e os passos do projeto de banco de dados. Do lado esquerdo está o processo *bottom-up* e do lado direito o processo *top-down*.

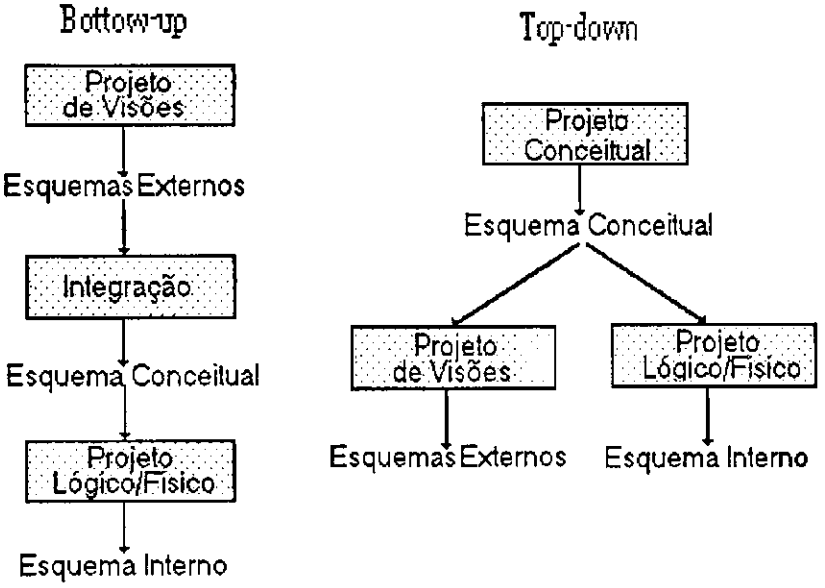

Figura 2.2 - Relacionamento ANSI/SPARC e processo de BD

A seguir definiremos os modelos de dados usados para definir o esquema conceitual. O papel da abstração nos modelos de dados é apresentar um modelo que forneça uma clara especificação dos dados e capture o significado dos dados. Essa abstração é medida pelo seu grau de expressividade. O modelo deve expressar nitidamente as diferenças entre tipos de dados, relacionamentos e restrições, e representar os relacionamentos *n* a *n* entre elementos.

2.2.1. Modelos de dados clássicos

Os modelos de dados hierárquico, rede e relacional são conhecidos por modelos de dados clássicos. As restrições que comprometem a expressividade desses modelos são: não existência de representação conceitual para alguns tipos de relacionamento; falta de suporte à generalização, agregação e agrupamento; inexistência de suporte à elementos com estrutura complexa.

Um banco de dados hierárquico é estruturado em árvore. Os nós da árvore são registros e a ligação entre os nós é conhecida como relacionamento *pai-filho*. A definição básica desse modelo é que um nó pai pode ter várias ocorrências de nós filhos, mas uma ocorrência do nó filho só pode pertencer a um único pai.

A estrutura do modelo em rede é fundamentada nos conceitos de arquivo de registros e interligações de registros. É uma estrutura mais geral do que o hierárquico. Uma determinada ocorrência de registros pode ter qualquer quantidade de superiores imediatos, bem como dependentes imediatos, possibilitando modelar os relacionamentos N:M. A maior desvantagem desse modelo é a complexidade, tanto na estrutura de dados, como na manipulação das informações suportadas pela lista encadeada.

Os modelos hierárquico e de rede não são flexíveis para modelar aplicações mais complexas, sendo mais voltados à implementação do que para suporte adequado à modelagem conceitual.

Nos anos 70, o modelo de dados relacional foi o grande marco na evolução dos modelos de bancos de dados. Esse modelo fornece mais independência dos dados por não considerar detalhes de implementação. Ele possui um embasamento teórico consistente e uma modelagem de dados simples, baseada nas noções de conjunto e relação. Mas a modelagem dos dados em tabelas leva a uma perda da semântica. Tal fato contribuiu para que surgissem outros modelos com um maior poder de expressividadesemântica.

2.2.2. Modelos semânticos

Os modelos semânticos, surgidos na segunda metade da década de 70, são os que melhor cobrem o objetivo de expressividade dos modelos de dados.

O mais conhecido modelo semântico é o modelo de entidade-relacionamento [Chen76]. A abstração de representação desse modelo é classificada da seguinte maneira: esse modelo tem somente 3 conceitos básicos (entidade, relacionamento e atributo), esses conceitos são formalizados com uma interpretação única e com uma notação diagramática simples. Tal modelo tem sido estendido com os conceitos de classe, subclasse, hierarquia baseada em generalização/especificação e, mais recentemente, regras. A idéia é enriquecer os modelos para dar suporte a todas as formas de abstrações de dados: generalização,

agregação e agrupamento.

No entanto, os modelos semânticos se mostraram incovenientes por serem apenas ferramentas de projeto conceitual de banco de dados, isto é, o esquema é projetado usando o modelo entidade-relacionamento e depois é mapeado para um esquema relacional.

Tentando dar também expressividade de comportamento às entidades, surge a idéia de acrescentar funções ao modelo semântico. A inclusão de relacionamentos funcionais dá origem aos modelos funcionais. A linguagem de dados Daplex [Shipman81] foi umas das primeiras linguagens suportadas pelo modelo de dados funcional. Ela fornece linguagem de definição de dados e de manipulação de dados baseada em funções. A principal vantagem desse modelo é o embasamento matemático das funções.

2.2.3. Modelos orientados a objetos

Os modelos de dados orientados a objetos são baseados nos conceitos dos modelos semânticos e das linguagens de programação. Dos modelos semânticos, eles mantêm as conexões de ocorrência e eliminam a distinção entre entidade e relacionamento. Das linguagens de programação, eles adquirem as características de encapsulamento, troca de mensagens e herança.

Os modelos de dados orientados a objetos sugiram para cobrir as deficiências dos sistemas de informações que vinham sendo contempladas pelo modelo relacional, quando diante de transações longas, manipulação de objetos complexos, necessidade de interface gráfica, e outras.

A utilização desses modelos possibilita um maior poder de expressão dos conceitos do mundo real, porque buscamos representar diretamente uma entidade do mundo real, qualquer que seja sua complexidade, por um único objeto do banco de dados. Não sendo necessário decompor a entidade em conceitos mais simples, como no caso dos modelos convencionais orientados a registros. Alcançando-se assim, uma representação mais natural.

O papel da abstração nos modelos de dados orientados a objetos é permitir uma modelagem conceitual mais modular e com encapsulamento de dados e métodos em tipos abstratos de dados.

As características gerais de um modelo orientado a objetos são: identidade de objeto, herança de propriedades e métodos, persistência, polimorfismo e amarração dinâmica. Maiores informações sobre os bancos de dados orientados a objetos serão vistas no próximo capítulo.

2.3. Abstração na área de engenharia de software

Existem dois mecanismos de abstração utilizados durante a construção de sistemas: abstração de procedimentos e de dados. A abstração de procedimentos é caracterizada pela utilização de uma função de um determinado nível abstrato, sem se preocupar com suas subfunções contidas num nível mais baixo. A abstração de dados é caracterizada pela utilização dos dados sem se preocupar com sua estrutura interna.

A abstração em engenharia de software pode ser vista em três perspectivas diferentes: em torno das linguagens de programação, das abordagens de desenvolvimento de sistemas e dos ambientes de desenvolvimento de sistemas.

2.3.1. Linguagem de programação

Durante toda a evolução das linguagens de programação tem sido objetivo dos projetistas incluir mecanismos para definir abstrações de dados e de controle. A abstração de controle se caracteriza pela utilização de módulos, subrotinas e estruturas de controle, como: se..faça, repita...até e enquanto...faça. Através da utilização desses mecanismos, permite-se que o programador se abstraia de detalhes de implementação, e se dedique mais à elaboração do algoritmo em si.

2.3.1.1. Linguagens de programação tradicionais

O evento que desencadeou o uso de abstração nas linguagens de programação foi em 1950, quando os programadores começaram a dar nomes mnemônicos às variáveis, ao invés de utilizarem valores em octal e se preocuparem com suas posições de memória.

Em 1960 surgiram Algol e Fortran suportando tipos de dados primitivos: inteiros, real, booleano e alguns mecanismos de estruturação bastante limitados (array) para dados homogêneos. Uma linguagem contemporânea a estas foi Cobol, que apresentou facilidades para agregação de dados em registros.

Com Simula-67 foi introduzido o conceito de objetos e classes. Os objetos tinham existência própria e podiam se comunicar uns com os outros. Já nesta época as operações foram chamadas de métodos [Khoshafian90].

Em 1970, com a crescente preocupação com a crise do software, houve a necessidade de construir programas estruturados. Os programas eram escritos, gradualmente, de forma *top-down*. Esta técnica possibilitava decompor problemas complexos em pequenos segmentos independentes, diminuindo a complexidade. Mas tal alternativa não satisfazia o programador. O programa final não preservava as abstrações utilizadas quando este foi criado, dificultando sua manutenção.

O suporte à abstração de dados teve maior impacto com os tipos enumerados de Pascal em 1975. Logo depois, com os tipos abstratos de dados das linguagens CLU e Módula [Shaw84]. A incorporação das idéias de tipos abstratos de dados em linguagens de programação levou a uma separação entre a definição de um tipo (especificação) e sua implementação. Ficando a especificação protegida de qualquer tentativa de manipulá-la diretamente. Qualquer mudança na implementação de um tipo abstrato de dado fica resumida ao trecho do programa, que o descreve, não afetando o resto do programa.

Esta nova definição requereu um novo enfoque do processo de programação, dando origem as linguagens de programação orientadas a objetos.

2.3.1.2. Linguagens de programação orientadas a objetos

A orientação a objeto representa para a informática nos anos 90, o que a programação estruturada representou nos anos 70: um importante paradigma para melhorar a criação, manutenção e utilização do software [Winblad93].

Um programa tradicional consiste de dados e procedimentos, um programa orientado a objetos consiste somente de objetos, que têm atributos específicos (dados) e maneiras de comportamento (métodos). As funcionalidades básicas das linguagens orientadas a objetos são: objetos com métodos e mensagens, hereditariedade, amarração dinâmica e polimorfismo.

Booch definiu alguns fatores essenciais que fazem uma linguagem ser verdadeiramente orientada a objetos e outros desejáveis de existirem [Norman92]:

- Classes para abstração - interfaces das classes são definidas independentemente de sua implementação.
- Classes com encapsulamento - um tipo de dado é definido por operações aplicáveis aos objetos desse tipo e esses objetos só podem ser modificados e consultados através dessas operações.
- Herança simples - na criação de uma subclasse, a partir de uma superclasse, são herdadas todas suas características.
- Objetos usando - um objeto pode fazer uso de serviços oferecidos por outros objetos;
- Objetos contendo - um objeto pode conter outros objetos como parte de sua estrutura.

Os fatores desejáveis definidos por Booch e que melhoram uma linguagem para usar o paradigma de orientação a objetos, mas não são necessários para qualificá-la como verdadeiramente orientada a objetos, são:

- Tipagem forte - as expressões são necessariamente de tipos consistentes.
- Tipagem dinâmica - os nomes são amarrados a tipos durante a execução.
- Polimorfismo - os subprogramas são distinguidos por seus argumentos.

O principal alvo dos que optam por uma linguagem orientada a objetos é conseguir desenvolver programas mais fáceis de modificar. A utilização dos conceitos de orientação a objetos auxilia esta manutenibilidade. O polimorfismo reduz o número de métodos e o tamanho do programa. A modularidade faz com que seja mais fácil conter os efeitos das alterações em um programa. A hereditariedade permite a criação de subclasses sem alterar um programa já escrito. A reusabilidade permite construir programas menos tendenciosos a erros, por utilizar classes já testadas de uma biblioteca de classes.

As características de orientação a objetos prometem oferecer um nível mais alto de abstração do que as suportadas pelas linguagens tradicionais. A decomposição do problema é feita em objetos, que são relativamente fáceis de definir, implementar e manter, porque refletem a modularidade natural de uma aplicação.

Podemos destacar as seguintes linguagens orientadas a objetos:

- ADA é uma linguagem bastante polêmica, quanto à sua classificação de ser ou não orientada a objeto. Segundo os fatores de Booch ela não é puramente orientada a objetos, porque não possui herança nem amarração dinâmica embora garanta encapsulamento dos dados.
- C++ teve boa receptividade no mercado por manter a portabilidade e eficiência da linguagem C e suportar os conceitos de orientação a objetos.
- Smalltalk é a linguagem que melhor suporta identidade de objetos, seguindo o princípio da homogeneidade: "tudo é objeto", incluindo classes e tipos.

2.3.2. Abordagens de desenvolvimento de sistemas

O conceito de desenvolvimento de sistemas foi estabelecido com o objetivo de identificar, organizar e compreender os requisitos do domínio de uma aplicação, facilitando sua implementação. Esta necessidade deu origem a elaboração de um ciclo de vida de desenvolvimento de sistemas. O termo ciclo de vida é normalmente usado para definir as fases do processo de desenvolvimento de software, especificando todas as atividades que devem ser desempenhadas nesse processo.

De uma forma geral, as fases do processo de desenvolvimento de sistemas são:

- **Análise**, que se inicia com um levantamento ou estudo de viabilidade do sistema, cuja tarefa é decidir se um projeto vale ou não a pena ser realizado. Quando o projeto é aprovado, o projetista deve especificar e avaliar as necessidades do usuário para definir as propriedades e funções que o sistema terá.
- **Projeto**, quando o projetista especifica *como* as necessidades do usuário devem ser implementadas.
- **Implementação**, que tem por objetivo gerar a estrutura dos dados e os programas preparados e testados. Ela é dedicada à construção de um sistema de produção do modo como foi especificado no projeto proposto, e numa verificação final de que o sistema construído executa a tarefa para a qual se destina.

Existem vários tipos de modelos de ciclo de vida, dentre eles podemos destacar:

- **Modelo de ciclo de vida clássico** que sugere uma abordagem sistemática e sequencial para o desenvolvimento de software, em que a passagem de uma fase para a próxima só é feita quando ela estiver totalmente terminada.
- **Modelo de ciclo de vida baseado em protótipos**, onde a construção de um sistema é feita utilizando protótipos para auxiliar na comunicação com o usuário. De forma que, desde a especificação de requisitos já se pode ir esboçando algumas características funcionais e de interface do sistema.
- **Modelo de ciclo de vida baseado em reutilização,** que consiste em gerar um novo sistema, incorporando qualquer produto já desenvolvido na especificação, no projeto ou na implementação de um outro sistema.
- **Modelo de ciclo de vida iterativo,** que se baseia em uma constante iteração entre todas fases do ciclo de vida de desenvolvimento de um sistema. Esse modelo é útil na abordagem de desenvolvimento orientada a objetos, porque existem classes que podem ser identificadas quando projetamos a solução, tendo que retornarmos a fase de análise para estudá-las no contexto, considerando também a possibilidade de reutilizá-las.

Existem várias abordagens que podem auxiliar no processo de desenvolvimento de sistemas, dentre elas consideramos: por função, por dados e por objetos [Capretz92]. Os fatores que diferenciam essas abordagens são como o problema deve ser visualizado no momento da construção do software e os mecanismos utilizados para a modelagem da solução do problema. A utilização da abstração nessas abordagens se apresenta quando o projetista vai fazer a decomposição de um problema e/ou representação das informações. Essas abordagens serão avaliadas a seguir, considerando as fases de análise e projeto.

2.3.2.1. Abordagem orientada a funções

O mecanismo de abstração que dá suporte nessa abordagem é de procedimentos, porque a ênfase funcional é bastante forte, pois os processos são decompostos em subprocessos e essa decomposição é bem representada nos diagramas de fluxo de dados.

O método de análise mais conhecido nessa abordagem é a análise estruturada. A filosofia considerada é mapear o mundo real seguindo os fluxos de dados existentes e todas as transformações sofridas por esses dados. Este mapeamento deve começar pelas características físicas do sistema atual, depois, derivar as características lógicas do sistema atual, construir um novo sistema lógico e, finalmente, incluir novas considerações físicas.

O método de projeto associado a análise estruturada é o projeto estruturado. Esse projeto objetiva sintetizar o processo do projeto fazendo uso das estratégias de análise de transformação e análise de transação, avaliar e melhorar a qualidade do projeto aplicando os critérios de coesão e acoplamento aos módulos.

Essa abordagem não é apropriada para aplicações em que há uma estrutura de dados hierárquica, dando espaço para que surgisse uma abordagem semanticamente mais representativa.

2.3.2.2 Abordagem orientada a estrutura dos dados

A utilização da abstração nessa abordagem se reflete quando o projetista parte da estrutura dos dados a serem processados para fazer a decomposição do problema, e quando ele utiliza diagramas hierárquicos para representar a estrutura de informação.

Os diversos métodos de análise que seguem essa abordagem, possuem as seguintes características comuns: identificação de objetos chaves do problema e, em seguida, as operações a serem executadas, representação destas informações em uma estrutura hierárquica, utilização de construtores elementares de sequência, seleção e repetição e mapeamento da estrutura de dados hierárquica em uma estrutura de programa.

As metodologias de projetos existentes nessa abordagem são utilizadas com o propósito de construir programas mais bem estruturados, dentre elas podemos destacar as de Jackson [Jackson75] e Warnier-Orr [Warnier77].

A metodologia de Jackson possui como idéia básica construir diagramas de estrutura, ordenados cronologicamente por ações das entidades, a fim de transformar os dados de entrada em saída.

A filosofia da metodologia de Warnier-Orr é fazer mapeamentos derivando a estrutura de dados e estrutura de programas, a partir dos elementos de saída.

Existem autores [Capretz92] que consideram o modelo de entidade-relacionamento [Chen76] orientado a estrutura de dados e outros [Yourdon92] que o coloca numa outra abordagem, denominada modelagem de informação. Na realidade, ambas enfatizam os dados antes que as funções e estruturam o sistema através dos dados e de seu significado. A diferença entre essas duas abordagens é sutil. O fundamento da estrutura de dados é: o problema é mapeado do mundo para o espaço das soluções a partir do detalhamento das estruturas de dados. Enquanto que o da modelagem de informação é: o problema é mapeado a partir do escopo dos objetos identificados do mundo real e dos relacionamentos estabelecidos entre esses objetos.

2.3.2.3. Abordagem orientada a objetos

A análise de sistemas orientados a objetos é a nova técnica que enfatiza a definição de características e do comportamento dentro de um sistema de objetos.

Nessa abordagem os mecanismos de abstração utilizados são de dados e de procedimentos. A

utilização de tipos abstratos de dados permite suportar a abstração de dados e a utilização de hierarquia de classes permite suportar a abstração de procedimentos. Dentro de uma hieraquia de classes, quando uma classe de alto nível requisita um método, ela se abstrai de que esse método pode enviar várias mensagens a outros objetos, e assim, sucessivamente.

O rápido desenvolvimento dessa abordagem pode ser atribuído a algumas razões: melhor poder de expressividade para modelagem das aplicações (o domínio do problema pode ser visualizado mais natural e realisticamente como uma coleção de objetos e métodos), melhor estrutura baseada no conceito de tipos abstratos de dados, possibilidade de reusar um software durante o desenvolvimento do sistema e continuidade conceitual, isto é, o enfoque da abstração é dado somente em torno de objetos, consequentemente, não é necessário mais nenhum mapeamento entre as fases do ciclo de vida, pois o projetista lida com os mesmos elementos do começo ao fim.

Podemos observar que, de uma forma geral, os passos de um processo orientado a objetos consistem de cinco atividades principais, que podem ser realizadas de uma forma não sequencial: identificação de objetos, organização de classes, identificação de operações, identificação dos relacionamentos entre objetos e elaboração de biblioteca de classes.

Dentre as várias técnicas que têm surgido propondo caminhos para identificar classes e métodos, encontramos as de verificação gramatical de documentos [Booch83], a de derivação do fluxo de dados e de diagramas de entidade-relacionamento, as heurísticas de [Yourdon92] e [Shlaer90]. Apesar de todo o esforço já realizado, chegou-se a um consenso de que para identificar classes e métodos não dá para seguir regras fixas, mas que devemos considerar algumas sugestões: modelar as entidades que aparecem no problema como classes, projetar métodos com um único objetivo, evitar métodos extensos, projetar um novo método quando tiver que ampliar um já existente, cada classe deve corresponder a uma abstração de dados, evitar hierarquias de classes extensas e com pouca ramificação.

2.3.3. Ambiente de desenvolvimento de sistemas

Um ambiente de desenvolvimento de sistema é composto de um ciclo de vida, um conjunto de diretivas para organizar o pensamento ao longo do processo de desenvolvimento (metodologias) e um conjunto de ferramentas [Mattoso89]. Seu objetivo é auxiliar os projetistas na elaboração de sistemas, pela integração de ferramentas e pelo gerenciamento das tarefas de administração associadas à programação.

A influência da abstração nesta área é livrar o projetista de trabalhos onerosos, garantir a consistência e a criação mais rápida de aplicativos.

Durante vários anos o suporte desses ambientes, se limitou as ferramentas de editor, compilador, linkeditor e depurador. Mas para suportar os novos conceitos de orientação a objetos, houve a

necessidade de ferramentas para controlar a definição, implementação e ligação de módulos e gerenciar bibliotecas de classes.

A utilização dessas interfaces de aplicativos faz surgir um novo estilo de diálogo, que abstrai o usuário de memorizar teclas associadas às funções. Ele é denominado manipulação direta [Monte 88].

A orientação a objetos influenciou a comunicação homem-máquina, baseada na manipulação direta, pelos seguintes motivos:

- O usuário emite comandos diretamente aos objetos visualizados.
- Cada objeto pode ser visto representando uma informação e executando as operações do usuário (já que a comunicação é feita diretamente a ele).
- O mesmo comando pode ser aplicado a objetos distintos, resultando em ações totalmente diferentes (polimorfismo).

Nos próximos anos, a tendência é construir ferramentas que deêm apoio à produção de sistemas multimídia, onde os programas dão respostas diversificadas com som e animação. A abstração será melhor representada , quando utilizarmos simulações de uma realidade. Esta nova técnica chama-se realidade visual, no qual o equipamento gera um ambiente áudio visual. As aplicações CAD orientadas a objetos já incluem esta capacidade de simulação, por exemplo, num projeto de engenharia, arquitetos e clientes caminham virtualmente dentro das construções simuladas por computador.

CAPÍTULO 3

Banco de dados orientados a objetos

3.1. Introdução

Nos anos 90, a idéia de gerenciamento de objeto tem sido o grande esforço das pesquisas , visando solucionar alguns problemas em que os bancos de dados tradicionais têm se mostrado inadequados.

Um dos problemas encontrados foi o crescente uso de aplicações de inteligência artificial, automação de escritório , engenharia (CAD, CAM e CASE), processamento de imagem, cartografia e multimídia. Essas aplicações passaram a solicitar um banco de dados que gerenciasse grandes quantidades de objetos complexos. Para expressar a semântica de objetos complexos são necessárias abstrações tanto de dados como de comportamento, o que não ocorre nos bancos de dados tradicionais.

Um outro problema é que nos sistemas de bancos de dados tradicionais, normalmente, as linguagens de definição de dados diferem da linguagem de programação utilizada para construção do restante da aplicação. Este fato é porque ambas possuem conceitos diferentes e sistemas de tipos diferentes e pouco compatíveis. Assim os programadores precisam aprender duas linguagens.

Além desses problemas, os bancos de dados tradicionais são frágeis quanto a extensibilidade de estrutura e de operações. Esse fato, força a criação de novas versões que levam a reorganizações do banco de dados, dificultando sua gerência.

Os bancos de dados orientados a objetos (BDOO) são capazes de dar suporte às aplicações complexas de forma mais efetiva do que os outros e solucionar o problema da compatibilidade das linguagens de definição e de manipulação dos dados. Eles, por representarem dados e operações, podem armazenar não somente componentes de aplicações complexas, mas também grandes estruturas de dados.

Nos BDOO, o problema da incompatibilidade das linguagens de definição de dados e de manipulação de dados pode ser resolvido com a extensão de uma linguagem de programação orientada a objetos para que ela seja utilizada como linguagem única de acesso ao banco de dados. Um exemplo desta estratégia é o sistema GemStone [Bretl89] que estendeu Smallalk para definir e manipular objetos no

3.3. Modelo de dados TOM

O modelo de dados TOM (Temporal Object Model) se propõe a fazer a modelagem conceitual do domínio da aplicação através dos aspectos estáticos e dinâmicos. Os aspectos estáticos se referem às estruturas dos dados em si. Os aspectos dinâmicos tratam da avaliação comportamental do objeto decorrente das suas mudanças de estado (evolução), em consequência da ocorrência de eventos.

O modelo de dados TOM é uma extensão do modelo semântico THM (Temporal-Hierarchic Model)[Schiel82] envolvendo características de orientação a objetos. Ele possui, além das facilidades de um modelo de objetos, regras de integridade associadas a cada métodos 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.

A seguir descreveremos o modelo TOM baseado nos conceitos de classe, metaclasse, relacionamento, abstrações, método, versão, restrição de integridade e tempo.

3.3.1. Classe

A criação de um banco de dados implica na definição das classes que farão parte dele. A sintaxe da definição de uma classe no TOM é vista na linguagem (tipo BNF) abaixo:

```
[Meta]class classe]
  [specialization of superclasse]
  [instance of metaclasse]
  [class relationships
   ( < nome rel> ":" < classe relacionada> "("
     < card min> ";" < card max> ")")
  (pre_class ":" < classe relacionada> [exclusive]) *
  (post_class ":" < classe relacionada> [exclusive]) * ]
  [instance relationships
   ( < nome rel> ":" < classe relacionada> "("
     < card min> "," < card max> ")")*
  [with < n> (history| rollback| temporal) values] )+ ]
  [class methods < def.método> + ]
  [instance methods < def.método> + ]
  (keys are (< lista de chaves+ > | inherited) |
  (type < tipo>                    [format < especificação> ])
  generalization of < lista de classes> [explict] |
    by predicate < condição>
    by role < nome papel> [parameters:[disjuntive][covering]]
  agregation [redefinition] of
    (< lista de classes> (all| explicit)|
    < nome classe-1, nome classe-2> by < relacionamento> ) [exclusive]
   grouping of < nome classe>
    (all | explicit | by predicate < condição> )
     [parameters: [disjuntive][covering][ordered] by < relacionamento> ]
  [parameters: with time [and lifetime < constante> : < unidade de tempo> ]]
```

A descrição de cada uma destas construções utilizadas na linguagem do modelo TOM pode ser encontrada em [David92].

3.3.2. Metaclasse

Uma metaclasse define objetos inerentes ao próprio modelo de dados e, portanto, independentes de qualquer aplicação. Elas correspondem ao nível do meta-esquema do modelo de dados, segundo a terminologia ANSI/SPARC [Mark85].

No modelo TOM, há dois níveis de abstração: o metanível e o nível de aplicação. O metanível é o nível composto por várias metaclasses, que descrevem a sintaxe e semântica desse modelo de dados. No metanível, o administrador de modelos pode criar ou modificar um modelo de dados específico para uma classe de aplicação, por criar novas metaclasses. Já existe a metaclasse pré-definida TOM_Class, que define a estrutura de uma classe no TOM. No nível de aplicação, o projetista cria classes de aplicação como instâncias ou subclasses das metaclasses do metanível, herdando destas propriedades temporais e de versionamento.

3.3.3. Relacionamento

Relacionamento são associações válidas entre dois objetos. Quanto à modelagem da semântica de um atributo de um objeto, alguns BDOO tratam o atributo de duas maneiras : Um atributo é também um objeto, ou um atributo é um valor de dado. Poucos BDOO têm uma clara distinção entre atributos que são dados e atributos que são referência a outros objetos. Nos modelos de dados semânticos, o sistema suporta atributos inversos na construção dos relacionamentos. Poucos sistemas orientados a objetos tem esta capacidade. O TOM trata relacionamentos como atributos e permite relacionamentos invertidos. Por exemplo, o relacionamento *TrabalhaPara*, especifica que *Firma* é atributo de *Empregado*, o relacionamento inverso é *PossuiEmpregado*.

A cada relacionamento é associado uma cardinalidade, expressa pelo par de valores (min, máx), que determina o mínimo e o máximo de instâncias associadas pelo relacionamento. Existem os seguintes tipos de relacionamentos:

3.3.3.1. Relacionamento instância

Relacionamento instância é aquele que varia de instância para instância. Por exemplo, o relacionamento chave *TemCódigo* relaciona objetos da classe *Obra* com os objetos da classe *CódigoObra*.

```
class Obra
      instance relationships
        TemCodigo: CodigoObra (1,1)
      keys are TemCodigo ...
```

3.3.3.2 Relacionamento classe

Relacionamento classe se refere à classe como um todo, não variando de instância para instância. Serve para considerar a classe como um objeto com seus próprios relacionamentos e para fatorar valores comuns a todas as instâncias. No exemplo abaixo, as propriedades *TemPeso* e *TotalPregos* pertencem à classe *Prego*.

```
class Prego
    class relationships
        TemPeso    : Peso (1,1)   /* relacionamento fatorado */
        TotalPregos: Inteiro      /* propriedade da classe */
```

3.3.4. Abstrações

As classes são organizadas numa estrutura hierárquica. Os relacionamentos hierárquicos entre classes podem ser considerados abstrações, no sentido de que detalhes de objetos nas classes inferiores são suprimidos na formação de classes de objetos mais gerais. Os relacionamentos hierárquicos representam as seguintes formas de abstração: generalização, agregação e agrupamento.

3.3.4.1. Especialização/Generalização

Especialização é o processo de selecionar de uma classe de objetos uma subclasse, dependendo de um critério de especialização, denominado papel. Esse papel exprime a condição que liga a subclasse à superclasse. Por exemplo, para o papel *CondiçãoTrabalho* obtemos as subclasses *Horista, Titular* de *Empregado*.

Generalização é o processo inverso da especialização. Esse processo é composto de 3 atividades: suprimir as diferenças entre algumas classes, identificar suas propriedades comuns e definir uma superclasse como generalização.

Existem 4 características de classificação dessa abstração:

- A generalização é dita **disjuntiva** quando as subclasses forem exclusivas. Por exemplo, uma *pessoa Feminina* não pode ser uma *pessoa Masculina,* mas ambas são especialização de *Pessoa*.
- A especialização é **completa** quando todos os objetos da superclasse ocorrem em pelo menos uma subclasse. Para o mesmo exemplo acima, podemos dizer que toda pessoa tem que ser ou do sexo masculino ou feminino.
- A generalização **explícita** ocorre quando ao incluirmos um objeto na superclasse, a correspondente subclasse for especificada explicitamente. No exemplo da classe *Empregado*, é necessário dizermos em qual condição de trabalho o empregado irá trabalhar.
- Generalização **por condição** é quando o critério de classificação dos objetos pode ser dado

por uma condição. Por exemplo, um *Funcionário* é considerado *MauPago* se seu salário for menor que X.

3.3.4.2 Agregação

Agregação é o processo de construir novos objetos a partir da composição de objetos existentes . O tipo de relacionamento que envolve as classes é denominado "parte de". Se as classes A e B forem parte da classe C, significa dizermos, que as instâncias da classe C são compostas por pares de elementos de A e B.

As características de classificação dessa abstração são:

- O parâmetro **todos** ocorre por considerarmos a classe agregada todo o produto cartesiano das classes componentes. Por exemplo, para a classe *Data* composta de *Dia, Mês* e *Ano*, qualquer valor do produto cartesiano aplicado à dia, mês e ano irá dar uma data.
- O parâmetro **explícito** possui a mesma semântica usada na generalização. No exemplo da classe *Casal*, e das classes componentes *Esposa* e *Marido*, deve ser especificado quem é casado com quem, não é qualquer combinação que é válida.
- A agregação **pelo relacionamento** ocorre quando o agregado é obtido do relacionamento entre as classes componentes. Na estrutura da figura 3.2, os componentes de uma *Aula* dependem do relacionamento *AssistemEm*, isto é, é formado pelos *Estudantes* que estão alocados a uma *Sala*.

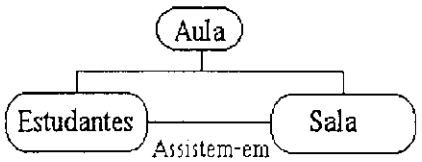

Figura 3.2 - Relacionamento Estudante/Sala

- A agregação é **exclusiva** quando um componente não pode participar em mais de uma agregação. Por exemplo, uma *CapaLivro* é parte de um único *Livro*.
- A agregação por **redefinição** consiste na redefinição de uma agregação que já existe a um nível mais alto de generalização. Por exemplo, uma classe B que é subclasse de A, pode redefinir os componentes da classe A, formando suas próprias classes componentes.

4.3.3.3. Agrupamento

Agrupamento é a forma de abstração que possibilita a definição de objetos de conjunto de instâncias de uma classe como instâncias de outra classe. O tipo de relacionamento considerado chama-se "é-elemento de". Se a classe A for elemento da classe B, significa dizermos, que as instâncias da classe B são conjuntos formados por instâncias da classe A. As características de classificação de um agrupamento são:

- Quando todos os elementos fazem parte do conjunto, o agrupamento é considerado **todos**. Por exemplo, a classe *ConjuntoDeInteiro* é o agrupamento de todos os *Inteiros*.
- Através do parâmetro **explícito** o usuário determina a estratégia de agrupamento. Por exemplo, a classe *TimeDeFutebol* de uma empresa é o agrupamento de 12 *Empregados*, sendo explicitamente especificado quem são eles.
- A característica **pela condição** tem a mesma semântica da generalização. Por exemplo, a classe grupo *FuncionárioParentes* é o agrupamento de todos os *Funcionários* que têm a

mesma família.

- Quando um objeto elemento não pode fazer parte de mais de um objeto grupo, o agrupamento é dito **disjuntivo**.
- Quando todo objeto elemento tem que estar associada a pelo menos a um objeto grupo, o agrupamento é **completo**.
- Os elementos podem ser **ordenados** de acordo com um relacionamento de instância especificado.

No TOM, herança de relacionamentos e de métodos são consideradas para as três abstrações. Para agregação e agrupamento é necessário especificar, para cada relacionamento e método, qual o tipo de herança envolvida:

- Se a herança for direta os relacionamentos e métodos são herdados diretamente pelo objeto de nível inferior sem modificações. Por exemplo, a classe *Automóvel* possui o relacionamento *TemDono* e o método *MudaDono*, que podem ser herdados pelas classes componentes *Motor, Porta* e *Chassis*.
- Quando a herança é computada os valores de algumas propriedades devem ser computados para serem herdados. Por exemplo, o método *IncluiCasal* pode ser a junção dos métodos *IncluiMarido* e *IncluiEsposa*.
- Existem casos em que não há herança nenhuma. Por exemplo, o relacionamento *TemValor* só se aplica a classe agregada *Livro*, e não a cada componente de um livro.

3.3.5. Método

Um método implementa um comportamento potencial dos objetos, que se reflete na alteração dos seus relacionamentos e envio de mensagens a outros objetos. O objeto que receber a mensagem responderá através da seleção e execução do método correspondente a mensagem solicitada, realizando um relacionamento dinâmico.

Existem dois tipos de métodos que podem aparecer na descrição de uma classe: método de instância se a mensagem for enviada a uma instância de uma classe ou método de classe se a mensagem for enviada a uma classe de objetos.

A estrutura geral de um método no TOM, obedece à seguinte sintaxe:

```
<nome método> ["(" <list a parâmetros entrada>")"]
  [pre-conditions
    [< condição> [otherwise (warning| cancel| error)]]*]
  [body
    < sequência de operações> ]
  [post-conditions
    [< condição> [otherwise (warning| cancel| error)]]*]
```

```
Metaclass EventoBD
  specialization of Evento
   instance relationships
    temNome: Promoção
    ativo        : Y
    método : PromoveFuncionário
    classe : Funcionário
```

```
Metaclass EventoTemporalPeriódico
    specialization of Evento
    instance relationships
     temNome: InícioMes
     ativo        : Y
     quando : < 93,01,00,00,00,00 >
     atraso :   < 00,00,00,00,00,00>
     período:  < 00,01,00,00,00,00 >
```

```
Metaclass Trigger
    instance relationships
       temNome : Adicional
       condição: Cargo.Funcionário =
"servidor"
       ação       : AdicioneUm
       f-ação     : AdicioneDois
```

[C] Regra

Regra é a associação entre eventos e *triggers*, determinando quais eventos irão ativar quais *triggers*. Após definir as regras, o campo *Regra* nas informações do evento, deve ser preenchido. Vejamos a seguir uma declaração da regra para disparar o *trigger Adicional* do exemplo acima:

```
Metaclass Regra
   instance relationship
    TemNome: RegraAdicional
    eventos: InícioMês, Promoção
    trigger: Adicional
```

Para que o *trigger* seja disparado é necessário que ocorra os eventos *InícioMês e Promoção*. Este é um caso típico de eventos concorrentes.

3.3.7. Versões

Versões são diversas ocorrências de um mesmo objeto, que coexistem numa base de dados e representam diferentes estados do mesmo objeto semântico [Dittrich 87]. O conceito de versão no TOM apresenta as seguintes características [Schiel92]:

- Referência 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 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 umas das outras).

Para representar uma classe versionada genérica na hierarquia de classes do modelo TOM, o usuário deve defini-la como subclasse da metaclasse G_CLASSE a fim de herdar o relacionamento *TemVersão* As versões individuais da classe genérica devem ser definidas como subclasse da metaclasse V_CLASSE para herdarem os relacionamentos *PréviaVersão* e *PróximaVersão*.

Vejamos a seguir como exemplo, um projeto de um sistema de software, composto de três fases: *Análise, Projeto* e *Programação.* As classes *Análise, Projeto* e *Programação* são versões individuais da classe *SistemaDeSoftware* No exemplo a seguir a classe *Projeto* possui como versão anterior a *Análise* e como versão posterior a *Programação*.

```
class Projeto
  specialization of V_CLASS
  instance relationships
   préviaVersão     : Análise (1,1)
   próximaVersão  : Programação>  (1,1)
```

3.3.8. Tempo

Um dos principais fundamentos do modelo TOM é a modelagem do tempo, sendo representado através das três opções abaixo:

3.3.8.1. Relacionamento temporal pré-pós

Quando um objeto eliminado de uma classe passa a pertencer a outra classe, diz-se que a primeira é a pré-classe (C1) e a segunda a pós-classe (C2). Neste relacionamento há três variações:

- Se todos os objetos que saem de C1 vão, obrigatoriamente, para C2, significa que a pós-classe é exclusiva;
- Se todos os objetos que estão em C2, vieram de C1 obrigatoriamente, se diz que a pré-classe é exclusiva;
- Se todos os objetos que saem de C1 vão para C2 e vice-versa, então é pré-classe exclusivo e pós-classe exclusivo.

Os relacionamentos pré-pós permitem estabelecer históricos da evolução de um objeto.

Consideremos a definição desses relacionamentos para a classe *Exemplar*, em que há uma dependência entre seus estados de *Emprestado* e *Perdido*.

```
class Emprestado
   class relationships
     pre_class: Disponível exclusive
     post_class: Disponível,Reservado,Perdido
```

```
class Perdido
   class relationships
     pre_class: Emprestado, Disponível
```

3.3.8.2. Relacionamentos com valores anteriores

Considerar a característica temporal no relacionamento de instância, significa que os valores dos relacionamentos alterados serão mantidos no banco de dados, para efeito de histórico, juntamente com a indicação das datas de início e término dos valores anteriores do relacionamento.

Considerando o relacionamento *TrabalhaDepto* entre as classes *Empregado* e *Departamento*, suponhamos ser necessário guardar todos os departamentos onde o empregado já trabalhou. Então na definição formal da classe *Empregado* abaixo há o histórico de até 10 departamentos em que o empregado foi alocado.

```
class Empregado
   instance relationships
    TrabalhaDepto : Departamento (1,*)  with 10 old values
```

3.3.8.3. Classes com tempo

Quando se diz que uma classe é temporal significa que os objetos removidos daquela classe permanecerão com um indicação do tempo de duração histórico do objeto na classe. Para evitar que as classes cresçam indefinidamente devemos especificar durante quanto tempo os objetos devem ser guardados.

3.3.8.4 Condição com Tempo

Quando se especifica uma condição temporal, significa que o predicado será considerado verdadeiro se numa determinada época ele era verdadeiro. Por exemplo, consideremos o predicado EstadoCivil.Empregado = "Casado" em 91:01:01:00:00:00.

Capítulo 4

Avaliação das metodologias de desenvolvimento orientadas a objetos

Uma metodologia deve ser utilizada para apoiar as atividades fundamentais de desenvolvimento de um sistema de computador. As metodologias são compostas de processo e/ou representação, onde processo refere-se aos procedimentos a serem seguidos nas fases do ciclo de vida do sistema, e representação refere-se à notação gráfica utilizada para especificar os resultados.

O objetivo deste capítulo é de fazer uma avaliação da qualidade das seguintes metodologias orientadas a objetos: Shlaer e Mellor[Shlaer90], Schiel[Schiel90], Rolland[Rolland91][Rolland89], Booch [Booch91] e Baptista[Baptista92]. Os motivos pelos quais essas metodologias foram escolhidas foi que as metodologias de Booch, Rolland e Schiel inspiraram a nova metodologia proposta, a metodologia de Shlaer e Mellor é um bom exemplo da combinação das características da análise tradicional com as da orientação a objetos e a metodologia de Baptista, porque contém o aperfeiçoamento do trabalho de Booch.

A estrutura deste capítulo está dividida da seguinte forma: primeiro, os critérios de avaliação são apresentados, depois, cada metodologia é avaliada e, por último, é apresentado uma tabela geral comparativa entre as metodologias avaliadas.

4.1. Critérios para avaliação da qualidade de uma metodologia

Os critérios utilizados para avaliação das metodologias orientadas a objetos, foram divididos em relação à três etapas distintas de apoio ao desenvolvimento de sistemas: construção do sistema, avaliação do sistema e gerência do processo de desenvolvimento. A figura 4.1 mostra esses critérios de avaliação relacionados a essas etapas. Esses critérios são baseados nos trabalhos de [Crispim91] e [Monarch92].

4.1.1. Construção do sistema

Um dos objetivos mais importantes de uma metodologia é auxiliar a construção de um sistema. Os

critérios de avaliação de uma metodologia utilizada durante a construção de um sistema são agrupados nos seguintes fatores: aplicabilidade, expressividade, facilidade de aprendizado e facilidade de uso.

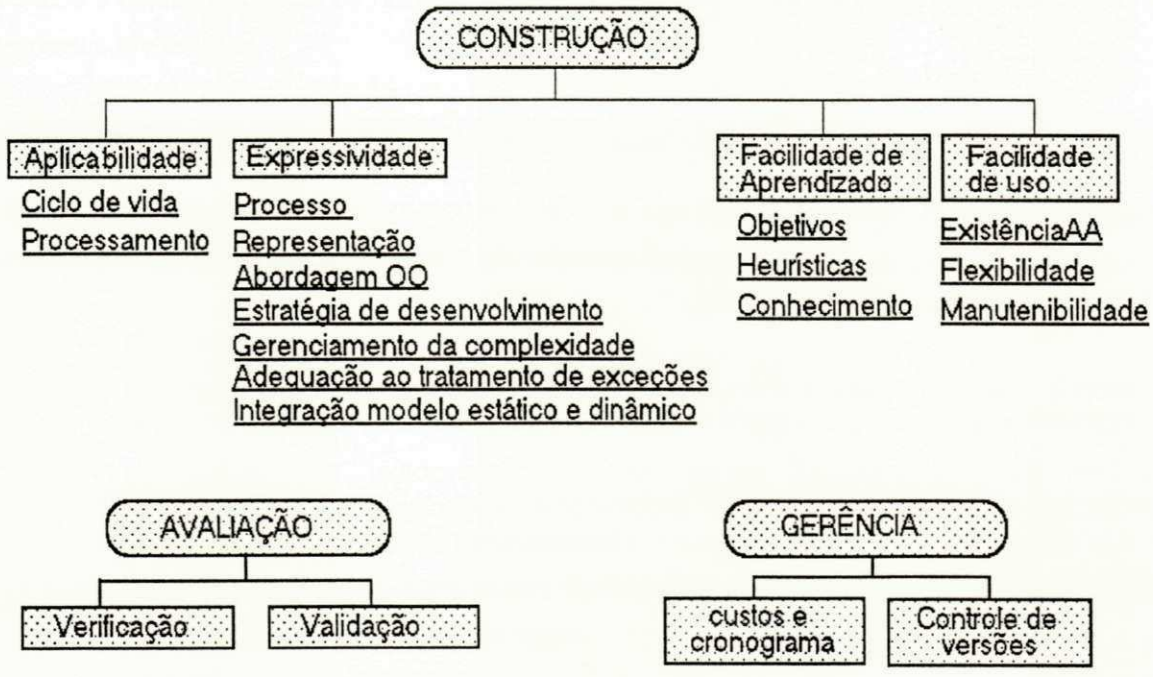

Figura 4.1 - Critérios de avaliação de qualidade

4.1.1.1. Aplicabilidade

Para escolhermos uma metodologia mais adequada é necessário avaliarmos sua aplicabilidade com relação às características do domínio da aplicação e ao contexto no qual o sistema será desenvolvido. Os critérios que avaliam a utilização da metodologia dizem respeito ao tipo de ciclo de vida e ao tipo de processamento que ela suporta.

a. Tipo de ciclo de vida

O ciclo de vida de um sistema diz respeito as atividades a serem realizadas durante as fases do processo de desenvolvimento de sistema. Os modelos de ciclo de vida considerados nesta análise e que foram devidamente definidos no capítulo 2, são: modelo de ciclo de vida clássico, modelo de ciclo de vida baseado em reutilização, baseado em protótipos e iterativo. Aqui, avaliamos a metodologia com relação a qual desses modelos suas características melhor se adequam.

b. Tipo de processamento

Com este critério avaliamos o tipo de processamento que a metodologia suporta. Os tipos de processamentos conhecidos são : lote, *on-line* e tempo real.

4.1.1.2. Expressividade

Os critérios que citaremos a seguir, servem para avaliar a expressividade de uma metodologia. Esses critérios são: processo e representação da metodologia, tipo da abordagem de desenvolvimento orientada a objeto, estratégia de desenvolvimento, gerenciamento da complexidade e adequação ao tratamento de exceções.

a. Processo

A avaliação feita quanto a este critério é determinar que passos das fases de análise e projeto a metodologia suporta. Os passos possíveis são descritos abaixo:

- Modelagem do fluxo das informações
- Identificação das classes (classes encontradas durante a fase de análise - classes do problema, classes encontradas durante o projeto - classes da solução, e classes temporais).
- Identificação dos relacionamentos entre classes (relacionamento de instância, relacionamento classe, relacionamento temporal, relacionamento dinâmico e abstrações - generalização, agregação e agrupamento)
- Identificação do comportamento das classes
- Identificação dos estados de um objeto
- Otimização de classes
- Identificação da concorrência dos métodos
- Definição das informações
- Integração do modelo estático e dinâmico

b. Representação da metodologia

Uma metodologia que é representada por uma notação expressiva, facilita a comunicação entre os participantes do desenvolvimento de um sistema. A avaliação feita quanto a este critério é determinar se a metodologia utiliza algum tipo de representação para apoiar os passos da metodologia.

c. Tipo da abordagem de desenvolvimento orientada a objetos

Este critério se refere ao tipo de abordagem de análise e projeto orientado a objeto utilizada para modelar um domínio. Existem três estratégias que caracterizam como é incorporado o paradigma de orientação a objetos às técnicas das metodologias:

- Na abordagem **combinativa** as técnicas tradicionais são combinadas com as técnicas de desenvolvimento orientadas a objetos. Geralmente, esta estratégia preserva as fases de especificação e de análise utilizando diagramas de fluxo de dados (DFD) e propõe heurísticas para converter esses diagramas em um modelo de objetos. O maior problema desta combinação é mapear os processos de DFD, os eventos e funções do modelo dinâmico para o comportamento de um objeto do modelo de objetos.

- A estratégia **adaptativa** trata-se de uma modificação nas técnicas tradicionais adaptando-as às técnicas de desenvolvimento orientadas a objetos. Por exemplo, os diagramas de fluxo de dados tradicionais podem ser adaptados para associar funções a objetos ou entidades.
- A estratégia chamada de **pura** usa somente a técnica orientada a objetos para modelar a estrutura, a funcionalidade e o comportamento dinâmico dos objetos.

d. Estratégia de desenvolvimento

Cada metodologia define a forma e a ordem na qual as decisões são efetuadas. As estratégias mais conhecidas são:

- **Top-down**, onde a partir de uma visão geral do sistema, ele é decomposto até chegar a um nível que torne compreensível todos os detalhes do sistema.
- **Bottom-up**, onde a partir da visão mais detalhada do sistema, ele é composto até obter abstrações do mais alto nível.
- **Round-trip gestalt**, que é uma mistura da técnica *top-down* (para análise e divisão do problema) e *bottom-up* (para manter e construir um sistema), onde os aspectos iterativos e incrementais são essenciais.
- **Most-critical-element-first**, onde as decisões que afetam as partes mais críticas do sistema são tomadas primeiro, fazendo com que os parâmetros críticos sejam todos satisfeitos.

e. Gerenciamento da complexidade

Geralmente, as metodologias controlam a complexidade da modelagem de sistemas grandes, dividindo a visão do domínio em vários níveis de abstração. A seguir mostramos os mecanismos que são levados em consideração para o particionamento do problema:

- Por **comunicação**, no qual os objetos que enviam mensagens para outros são colocados no mesmo nível de abstração;
- Por **estado**, onde os estados de um objeto que são afetados por uma determinada função são nivelados [Baptista92];
- Por **assunto**, onde aqueles objetos que se encontram no nível mais alto de abstração de um modelo de estrutura ou aqueles não pertencentes à nenhuma estrutura são colocados todos num mesmo nível [Yourdon90];
- Por **subsistema**, onde o nivelamento é feito com os objetos que representam as funções do sistema, por exemplo, um subsistema financeiro e outro de relações humanas;
- Por **categoria de classes** no qual as classes da mesma semântica são agrupadas formando categoria de classes [Booch91].

f. Adequação ao tratamento de exceções

Devido a ocorrência de situações que fogem as regras previamente definidas, devemos nos prevenir representando o tratamento de exceções no desenvolvimento do sistema. Este critério avalia se as características inerentes a metodologia possuem tratamento de exceções.

4.1.1.3. Facilidade de aprendizado para construção

Este fator se refere aos recursos disponíveis ao usuário, que o auxiliam no aprendizado da metodologia.

a. Clareza de objetivos

Este critério se refere a qualidade da definição dos passos, que descrevem a metodologia. É necessário que fique bem claro, *o que fazer, como fazer e qual* o produto final de cada passo.

b. Existência de heurísticas

Heurísticas são regras previamente definidas, que servem como guia a um projetista, quando ele pretende realizar uma atividade de construção do sistema. Neste critério vemos que tipo de heurísticas são fornecidas pela metodologia.

c. Exigência de conhecimento para construção

Este critério avalia o conhecimento requisitado do projetista, para ele utilizar a metodologia.

4.1.1.4. Facilidade de uso

Os critérios deste item estão relacionados a facilidade de desenvolver e alterar um sistema em desenvolvimento e ao grau de automação da metodologia.

a. Existência de apoio automatizado

Para evitar o trabalho tedioso de desenhar gráficos, diagramas e fazer consistências manualmente, verificamos a existência de alguma ferramenta de apoio automatizada (AA) à metodologia.

b. Manutenibilidade

Neste critério avaliamos como a metodologia se comporta para modificar a especificação do sistema, diante de qualquer alteração do ambiente e da correção de erros.

c. Flexibilidade

Neste critério verificamos se o processo da metodologia é flexível. Uma metodologia é dita flexível quando os passos que a descrevem podem ser seguidos de forma paralela, e não somente sequenciais.

4.1.2. Avaliação do sistema

Após construírmos um sistema queremos um produto de boa qualidade e que atenda às necessidades do usuário. Existem técnicas que permitem avaliar o sistema construído, como: técnicas de verificação

e validação.

4.1.2.1. Verificação

Em relação a este critério buscamos descobrir se a metodologia provém uma maneira para verificar se os resultados obtidos em uma fase são consistentes com os da fase anterior. A especificação de requisitos, inspeções, revisões são os recursos utilizados para avaliar a qualidade dos produtos gerados e para resolver problemas de conflito, omissões e ambiguidades.

4.1.2.2. Validação

Validação é o processo de verificar se o sistema proposto está de acordo com as necessidades do usuário. Os recursos mais conhecidos para auxiliar esse processo são regras que testem se não há contradições, se o sistema está completo e se as normas impostas pelo modelo conceitual são respeitadas e protótipos. A avaliação feita com relação a este critério é determinar que recursos a metodologia utiliza para validar o sistema gerado.

4.1.3. Gerência do sistema

As tarefas da gerência de um sistema estão relacionadas ao planejamento e acompanhamento do projeto, controlando os recursos monetários, o tempo de desenvolvimento para o sistema e as possíveis alternativas de projeto.

4.1.3.1. Suporte para acompanhamento de custos e de cronogramas

Este suporte se refere a existência de técnicas que acompanhem os custos de desenvolvimento de um sistema e elaborem cronogramas das atividades a serem realizadas.

4.1.3.2. Suporte para controle de versões

Este critério serve para avaliar se uma metodologia provém formas de representar várias versões do software, que está sendo desenvolvido e de como ela auxilia nas decisões gerenciais.

4.2. Descrição e avaliação das metodologias

O processo de apresentação de cada metodologia é feito em 3 etapas. Na primeira etapa, fazemos a descrição da metodologia contendo os seguintes itens: nome, autor, ano em que a metodologia foi desenvolvida, sua filosofia, representação e processo. Na segunda etapa, construímos uma tabela

especificando as características da metodologia com relação aos critérios de avaliação de qualidade considerados. Na última etapa, a partir da tabela construída, fazemos uma análise crítica, buscando identificar as vantagens e desvantagens da metodologia.

4.2.1. Metodologia "Análise de sistemas orientada a objetos"

4.2.1.1. Descrição da metodologia

Esta metodologia foi desenvolvida por Sally Shlaer e Stephen Mellor em 1988. É uma metodologia de análise, e possui um sistema de notação híbrida para a combinação de análise estruturada com técnicas de orientação a objetos e um padrão de documentação para projetos de sistemas orientados a objetos. A metodologia adota o conceito de objeto como um registro em um banco de dados e fornece estratégias de análise de dados para modelar os dados.

Os recursos utilizados para apoiar a análise orientada a objetos são:

- **Diagrama da Estrutura da Informação** (DEI), que é utilizado para identificar os objetos do domínio do problema, atributos e relacionamentos. Contém a mesma filosofia do diagrama de entidade-relacionamento sendo nele representado também os relacionamentos inversos e a cardinalidade.
- **Diagrama de Contexto da Estrutura da Informação** (DCEI) contendo a mesma estrutura e simbologia do DEI, e que objetiva unir todas as estruturas dando um visão geral do contexto.
- **Diagrama de Transição de Estados** (DTE) que descreve todos os eventos, estados e atividades para cada objeto. As atividades são associadas com os estados, de forma que na entrada de um estado, todas as atividades são solicitadas a ocorrer.
- **Diagrama de Fluxo de Dados** (DFD) que é o recurso utilizado na análise estruturada para modelar fluxos de dados de um sistema. Nesse diagrama são representadas as atividades do diagrama de transição de estado.
- **Especificação de processo** (EP), recurso que teve origem no projeto estruturado, cuja finalidade é especificar os processos. As técnicas utilizadas são: português estruturado, árvores de decisão ou pseudo-código.
- **Documento de Especificação de Objeto** (DEO) que objetiva descrever os objetos e atributos do domínio, através de uma relação de todos os objetos e da especificação de cada objeto no modelo.
- **Documento de Especificação de Relacionamento** (DER), o qual é utilizado para identificar e descrever os relacionamentos de cada objeto e sua cardinalidade.
- **Documento de Resumo** (DR), o qual objetiva resumir as informações dos objetos, seus atributos e relacionamentos, contidas nos DEO e DER.

Esta metodologia apresenta 4 fases de processo de desenvolvimento de sistema: análise do problema, especificação externa, projeto de sistema e implementação.

Análise do problema

O processo desta metodologia é composto por oito passos, nos quais são agrupados em duas fases de desenvolvimento: análise e projeto.

Fase de Análise

Desenvolvimento do grafo comportamental: Este grafo descreve, nos diversos níveis de abstração, os aspectos dinâmicos da aplicação, seguindo a abordagem de decomposição funcional.

Derivação do gráfico de estrutura: a partir da descrição comportamental da aplicação representada no grafo comportamental, podemos extrair a estrutura de cada objeto e seus relacionamentos, construindo o grafo de estrutura. Uma unidade de informação do GC pode corresponder a um objeto ou um relacionamento do GE.

Análise temporal: o objetivo de modelar o tempo é de permitir que realizemos alterações e consultas antes ou depois de um determinado tempo. O produto deste passo é construir uma tabela de tempo e acesso.

Análise de restrições: neste passo todas as condições e conhecimentos sobre a aplicação, são colocadas na tabela de condições e tabela de regras.

Fase de projeto

Esta fase trata-se do projeto dos requisitos e da descrição do projeto para a linguagem VML. Essa é a linguagem conceitual para o modelo de dados VODAK.

Projeto em Petri-Net: neste passo devemos fazer um projeto em Petri-Net com somente as informações que devem ser colocadas no computador. As informações do grafo comportamental, as entradas da tabela de tempo e acesso, as restrições da tabela de condições pré-pós e as regras na forma de evento, condição, ação são convertidas nas notações utilizadas na rede de Petri. Existem algumas heurísticas que determinam como esta conversão deve ser feita.

Projeto de esquema de objeto: o objetivo deste passo é classificar as entidades contidas no grafo estrutural das informações, gerando outro grafo de estrutura. Assim, nesse novo grafo de estrutura, nem todos os nós são entidades, por exemplo, um nó pode ser uma classe de objeto ou um valor. A metodologia sugere algumas heurísticas para a classificação das entidades e para a derivação dos relacionamentos hierárquicos.

Aplicação das abstrações: a partir da rede de Petri, as informações comportamentais de um objeto devem ser encapsuladas numa classe de objeto, gerando um esquema de objetos. Os relacionamentos entre as classes de objetos é feita através da chamada dos métodos.

Conversão para VML: após a definição do modelo de objetos, o esquema é convertido para a sintaxe VML.

4.2.2.2. Avaliação da metodologia

Consideremos a tabela da figura 4.4 contendo a avaliação desta metodologia.

4.2.2.3. Análise crítica

Embora esta metodologia apresente um bom suporte às diferentes formas de abstração, tanto a nível estrutural como a nível comportamental, ela é fundamentada na decomposição funcional. Funções são propriedades dinâmicas, não sendo confiável desenvolver um projeto, tomando como base propriedades mutáveis.

Quanto ao gerenciamento da complexidade, vimos que a metodologia apresenta uma abordagem para controlar o crescimento dos grafos comportamentais (decomposição funcional), mas não existe nenhum controle de complexidade dos grafos estruturais.

Alguns erros são encontrados somente no momento de classificar as entidades ou no encapsulamento das informações. Para um processo sequencial isto significa mudanças desde o primeiro grafo comportamental. Tal trabalho é complexo e tedioso.

Não existe suporte à otimização de classes. É necessário elevar as informações estruturais e comportamentais comuns para as classes mais abstratas, otimizando a hierarquia de classes e melhorando a qualidade das classes a serem reutilizáveis.

4.2.3. Metodologia "Modelisation conceptuelle orientée objet"

4.2.3.1. Descrição da metodologia

Esta metodologia foi desenvolvida por Rolland em 1990 [Rolland90] e modificada em 1991 quanto às características comportamentais [Rolland 91]. É uma metodologia orientada a objetos para um banco de dados orientado a objetos, combinando as vantagens do modelo semântico com os conceitos de orientação a objetos.

O banco de dados é considerado como um conjunto de objetos, que interagem em consequência da ocorrência de eventos. O esquema de banco de dados é também visto como uma coleção de objetos incluindo objetos estáticos (atributos e eventos), objetos dinâmicos (operações e as condições de ocorrência de eventos) e objetos comportamentais (gatilhos e dependência de eventos).

A representação utilizada para auxiliar o processo de desenvolvimento de um sistema é constituída de

CONSTRUÇÃO DO SISTEMA — Schiel

APLICAÇÃO

Ciclo de vida	Clássico
Processamento	Tempo real

EXPRESSÃO

Processo:	Representação:
Modelagem do fluxo das informações	grafo comportamental
Identificação de classes:	
Classes do problema	grafo estrutural
Classes da solução	esquema estrutural TOM
Classes temporais	grafo estrutural
Identificação dos relacionamentos:	
Relacionamento Instância (atributo)	grafo estrutural
Relacionamento classe	grafo estrutural
Relacionamento temporal	sim
Relacionamento dinâmico (troca mensagem)	esquema de objeto
Generalização	esquema estrutural TOM
Agregação	esquema estrutural TOM
Agrupamento	esquema estrutural TOM
Identificação do comportamento	esquema de objeto e rede PrT
Identificação dos estados de objeto	sim
Otimização de classes	não
Identificação de concorrência	rede PrT
Definição das informações	tabelas: condição,regra e tempo
Integração modelo estático e dinâmico	esquema de objeto
Abordagem OO	combinativa
Estratégia de desenvolvimento	*gestalt*
Gerenciamento da complexidade	função
Adequação ao tratamento de exceções	sim

FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO

Clareza de objetivos - Sim

Existência de heurísticas - Transição de Petri-Net para o modelo de objetos

Exigência de conhecimento - AE, OO e Redes de Petri

FACILIDADE DE USO

Existência de AA - Em fase de implementação

Flexibilidade - As fases do processo são sequenciais, não permitindo uma iteração de informações para adquirir maior precisão.

Manutenibilidade - Uma mudança na especificação de um sistema projetado numa abordagem de decomposição funcional, resulta numa mudança desde o topo do projeto até o mais detalhado projeto.

AVALIAÇÃO DO SISTEMA

VERIFICAÇÃO

Não

VALIDAÇÃO

O método utiliza linguagem gráfica, mas a conversão para Petri-Net é complicada. Como alguns erros contido nos altos níveis de abstração, só são detectados na conversão das entidades para objetos e aplicação da abstração, podemos construir um software que não é o que o usuário quer.

GERÊNCIA DO SISTEMA

ACOMPANHAMENTO DE CUSTOS E CRONOGRAMA

Não

CONTROLE DE VERSÕES

Não

Figura 4.4 - Tabela de avaliação de Schiel

4 diagramas:

- **Esquemaestático**, que permite representar os objetos e os relacionamentos estruturais das duas formas de abstração: agregação e agrupamento.
- **Esquemahierárquico**, que é utilizado para representar a generalização.
- **Gráfico de ciclo de vida ,** onde é representado a evolução do ciclo de vida de um objeto, ele é comumente conhecido por diagrama de transição de estados.
- **Gráfico de dependência,** nele são representados a ocorrência de eventos em 3 tipos possíveis de interação entre os objetos: compartilhamento, disparo e referência. O modelo é mostrado na figura 4.5.
  - » Compartilhamento - a interação por compartilhamento exprime os pontos de sincronização da vida dos objetos. É causada pela ocorrência de eventos que se produzem, simultaneamente, no ciclo de vida de 2 objetos. Existe uma dependência temporal entre os objetos.
  - » Disparo - um evento pode disparar ações nele mesmo ou em outros objetos.

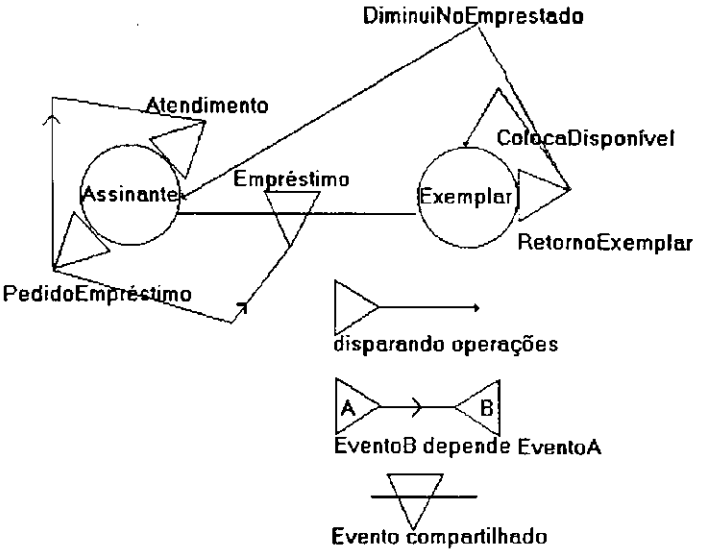

figura 4.5 - Gráfico de dependência

  - » Referência - exprime uma dependência cronológica entre eventos. Por exemplo, um evento A só acontece após a ocorrência dos eventos B e C.

O processo dessa metodologia é formado por 6 passos iterativos e não sequenciais.

Identificação dos objetos do mundo real, onde é feita a identificação dos objetos e também dos eventos.

Descrição dos objetos, onde são descritas às propriedades dos objetos (atributos e estados), que fornecem as informações da evolução dos objetos. O produto deste passo é fornecer uma lista de atributos, estados e eventos.

Análise dos objetos, aqui, de posse das informações de estado e eventos, já se pode construir o gráfico de ciclo de vida. Analisando os eventos que entram e saem dos estados contidos nesse gráfico, determina-se a condição de ocorrência dos eventos e completa-se a parte da definição de eventos no

esquema de objetos. As operações oferecidas e utilizadas pelos objetos podem ser identificadas, e a parte da definição de operações no esquema de objetos pode ser completada.

Organização da estrutura dos objetos, onde são analisados os objetos e seus relacionamentos para definirmos a estrutura da base de dados. É aconselhável aplicar a generalização objetivando diminuir a redundância de especificação e armazenamento.

Organização do comportamento dos objetos, neste momento, os eventos já encontrados devem ser analisados, considerando as três formas de interação entre os objetos: compartilhamento, disparo e referência. O produto deste passo é construir o gráfico de dependência e completar o esquema de objetos.

Especificação do esquema de objetos, onde o esquema conceitual final é obtido por juntar todos os esquema de objetos, definidos durante as etapas anteriores. Esse esquema conceitual deve ser validado por aplicar regras de conformidade (assegura as normas do modelo conceitual), de coerência (verifica a existência de incoerências), de completude (assegura a definição de todo elemento) e de fidelidade (verifica se o esquema conceitual é fiel à requisição do usuário).

4.2.3.2. Avaliação da metodologia

Consideremos a tabela da figura 4.6 contendo a avaliação desta metodologia.

4.2.3.3. Análise crítica

Esta metodologia aborda os conceitos de bancos de dados e tratamento de eventos, representando de uma forma clara a dependência entre os objetos. Isto justifica porque a metodologia FADO adotou algumas idéias dela.

A inexistência de protótipos não ajuda a validar o projeto, mas a metodologia utiliza como alternativa a aplicação de regras. Quanto à representação podemos dizer que não há necessidade de dois diagramas estruturais: estático e hierárquico, eles podem ser integrados num só. Observando o gráfico de dependência, pudemos perceber que existem alguns pontos ausentes na representação e que dificultam o trabalho do projetista, como: que gatilho está associado a ação disparada, a ordem de execução dos gatilhos, que método gerou o evento, quais parâmetros estão associados aos métodos.

A maneira utilizada para expressar métodos ou mensagens entre objetos, não dá clareza como métodos e atributos são encapsulados com os objetos.

4.2.4. Metodologia "Object-Oriented Design"

4.2.4.1. Descrição da metodologia

CONSTRUÇÃO DO SISTEMA: Rolland

APLICAÇÃO

Ciclo de vida	Iterativo sem protótipo
Processamento	*on-line*

EXPRESSÃO

Processo:	Representação:
Modelagem do fluxo das informações	não
Identificação de classes :	
Classes do problema	sim
Classes da solução	não
Classes temporais	não
Identificação dos relacionamentos:	
Relacionamento Instância (atributo)	esquema estático
Relacionamento classe	não
Relacionamento temporal	não
Relacionamento dinâmico (troca mensagem)	grafo de dependência
Generalização	esquema hierárquico
Agregação	esquema estático
Agrupamento	esquema estático
Identificação do comportamento	grafo de dependência
Identificação dos estados de objeto	grafo de ciclo de vida
Otimização de classes	sim
Identificação de concorrência	grafo de dependência
Definição das informações	diretamente da estrutura de BD
Integração modelo estático e dinâmico	não
Abordagem OO	puramente OO
Estratégia de desenvolvimento	*top-down*
Gerenciamento da complexidade	comunicação
Adequação ao tratamento de exceções	não

FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO

Clareza de objetivos - Sim
Existência de heurísticas - Otimização da estrutura das classes
Exigência de conhecimento - Conhecer conceitos de OO

FACILIDADE DE USO

Existência de AA - Sim
Flexibilidade - Somente os passos 4 e 5 são examinados em paralelo
Manutenibilidade - As alterações são facilitadas porque a mudança se limita ao objeto afetado, considerando que um objeto é independente de outro.

AVALIAÇÃO DO SISTEMA

VERIFICAÇÃO

Não

VALIDAÇÃO

A metodologia fornece regras para validar o esquema conceitual produzido, A compreensão dos modelos dinâmicos gerados é dificultada porque não existe integração com os modelos estáticos, nem dar idéia de objeto encapsulado.

GERÊNCIA DO SISTEMA

ACOMPANHAMENTO DE CUSTOS E CRONOGRAMA

Não

CONTROLE DE VERSÕES

Não

Figura 4.6 - Tabela de avaliação de Rolland

Esta metodologia foi desenvolvida por Grady Booch em 1991, sendo bastante conhecida hoje em dia. Ela é uma metodologia de projeto, baseada na decomposição de objetos, isto é, o sistema é decomposto diretamente das classes que fazem parte do domínio do problema. de uma forma incremental e iterativa.

A metodologia possui 6 diagramas principais, dentre eles: diagrama de classe e diagrama de objeto para representar projeto lógico, diagrama de módulo e diagrama de processo para projeto físico e o diagrama de transição de estado e diagrama de tempo para representar a visão semântica estática e dinâmica, respectivamente. Para sistemas complexos há a necessidade do diagrama de categoria de classes, que se decompõe em vários diagramas de classes e do diagrama de subsistemas, que se decompõe em vários diagramas de módulos.

A seguir descreveremos cada um desses diagramas considerando os relacionamentos e propriedades representados:

- **Diagramadeclasses** que mostra a existência das classes, seus relacionamentos, bem como a cardinalidade, visibilidade, concorrência e persistência das classes. Os diagramas de classes são organizados de forma parcialmente hierárquica, dentro do mesmo diagrama existem classes de níveis diferentes, onde as de nível mais alto estão posicionadas na parte superior do diagrama. Um dos motivos dos diagramas não serem totalmente hierárquicos, comumente conhecido pela técnica "top-down", é porque quando esta é usada, ela é associada à decomposição funcional das partições do sistema, e não à decomposição de objetos. Outro motivo é que essa técnica não é apropriada para reusabilidade, isto é, incorporar classes já definidas e validadas da biblioteca. Neste caso, a técnica mais adequada é "bottom-up", pois se começa por uma classe já detalhada. Porém "bottom-up" também não é a intenção, o ideal é a combinação das duas técnicas, conhecido por *round-trip gestalt*. Pois há casos em que, estando num nível X procurando classes que são parte do problema, queremos fazer uso de uma classe utilitária. Como as classes utilitárias são primitivas, é necessário desviarmos à um nível de abstração mais baixo. Após este desvio, voltamos ao nível X para continuarmos a especificação.
- **Diagrama de categoria de classes**, usado para agrupar em categorias, classes com propriedades comuns. Na prática um sistema médio tem um diagrama de classe de alto nível consistindo de classes, e um sistema grande tem um diagrama de categoria de classes, onde cada categoria possui um diagrama de classe associado.
- **Diagrama de objetos**, que representa a invocação de um método através de troca de mensagens entre os objetos. As propriedades representadas nesse diagrama são de concorrência, sincronização de mensagens, visibilidade e persistência. Os relacionamentos entre objetos abrangem que tipo de declarações cada um faz sobre o outro, quais as operações que podem ser realizadas e qual o comportamento resultante. Os tipos de relacionamentos são: relacionamento *usando* (quando um objeto pode fazer uso de serviços oferecidos por outros objetos) ou relacionamento *contendo* (quando um objeto pode conter outros objetos como parte de sua estrutura). O diagrama de objetos da figura 4.7 mostra como

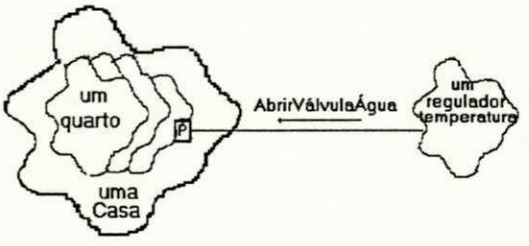

Figura 4.7 - Diagrama de objeto

representar um objeto agregado.

- **Diagrama de módulo** que é usado para fazer a alocação de classes e objetos em módulos, na visão física do sistema, e tem por objetivo agrupar abstrações e dar enfoque a decisões de visibilidade.
- **Diagrama de subsistemas** que é utilizado para representar subsistemas que são agrupadores de módulos. O objetivo desse diagrama é facilitar o manuseio de sistemas com grande quantidade de diagramas de módulo.
- **Diagrama de processos** que é usado para fazer a alocação de processos (programas) em processadores, como parte do projeto físico do sistema.
- **Diagrama de transição de estados,**que representa o comportamento dinâmico das instâncias das classes. Nele descrevemos os diferentes estados dos objetos e os eventos e/ou ações que causam a transição de um estado para outro.
- **Diagrama de tempo** que é utilizado para controlar as atividades de tempo crítico. Isso foi idealizado po Booch porque o diagrama de objetos não representava o momento exato da ocorrência das operações. O diagrama de tempo representa tanto o fluxo de controle como a ordem dos eventos. Através dele percebemos o início e fim das operações, que seguem entre os vários objetos, e o momento da criação e destruição dos objetos.

O objetivo básico dessa metodologia é manter a visão lógica , representada pela estrutura de classes e objetos, relacionada à visão física, representada pela arquitetura de módulos e de processos. A seguir apresentamos as 4 fases do seu processo.

Identificação de classes e objetos, onde são definidos os limites entre o sistema e o resto do mundo real, para definir os objetos que interagem no escopo do problema.

Identificação da semântica de classes e objetos, aqui, identificar a semântica é descobrir o significado de cada classe chave. Booch sugere escrever um texto para cada objeto, especificando o que o objeto faz e o que ele requer. Nesta fase devemos refinar os *templates*, isto é, fornecer a documentação da semântica estática e dinâmica das classes. A seguir definimos o meio utilizado para documentar cada semântica:

- Semântica estática das classes - *template* de classes
- Semântica estática de objetos - *template* de objetos
- Semântica estática dos métodos - *template* de operação
- Semântica dinâmica de Cl e obj - Diagrama estado transição
- Semântica dinâmica dos métodos - Diagrama de tempo

Identificação dos relacionamentos entre classes e objetos, neste momento deve-se, inicialmente estabelecer os relacionamentos possíveis entre classes e objetos, considerando a visibilidade entre eles. Depois deve-se projetar os diagramas de módulos considerando alguma decisão de visibilidade, que já se tenha tomado. Se forem reconhecidas partes comuns entre objetos, pode-se tomar decisões sobre módulos, isto é, dependendo da semelhança, as classes podem ser colocadas no mesmo módulo.

Implementação de classes e objetos, aqui, é estabelecida a concreta interface de cada classe e são tomadas as decisões de projeto físico, tais como: alocar a representação de cada classe e objeto a módulos, e de alocar programas a processos.

4.2.4.2. Avaliação da metodologia

Consideremos a tabela da figura 4.8 contendo a avaliação dessa metodologia.

4.2.4.3. Análise crítica

A metodologia de Booch é mais voltada para projeto orientado a objetos, onde foi validada em cinco linguagens diferentes: C++, Ada, Smalltalk, Pascal e Lisp. Existe independência de linguagem, dando possibilidade ao projetista de fazer sua decisão de projeto arquitetural baseado na linguagem escolhida. O que não foi o caso das versões anteriores [Booch83][Booch86] em que eram, exclusivamente, dependentes de Ada.

Quanto a representação, a metodologia apresenta muitos diagramas (8), exigindo do projetista saber tantas notações e compromentendo a concisão na especificação do sistema.

Em geral, a notação gráfica dos diagramas é clara, mas a do diagrama de objetos é muito confusa. Existem várias maneiras de representarmos a troca de mensagens entre os objetos e de como desenharmos os objetos (dentro de um objeto maior, os objetos podem vir próximos um do outro ou um aninhamento de objetos). Se esse tipo de representação fica mais próximo da implementação, por outro lado faz o projetista se preocupar com detalhes da estrutura dos dados, que deveriam ser adiados o máximo possível.

O diagrama de objetos representa a troca de mensagens entre objetos, porém esse diagrama possui característica estática, isto é, não representa os eventos , que acontecem dinamicamente no sistema. Se os eventos fossem representados, a ordem de dependência cronológica entre eles poderia ser controlada sem a necessidade de ter um diagrama de tempo para tal.

A definição do processo desta metodologia é feita de uma forma bem resumida. Além disto, ela não está didaticamente bem definida. As definições são espalhadas no livro, não possibilitando ao leitor uma sequência de raciocínio.

Quanto a adequação ao tratamento de exceções só existe no momento de especificar o método no *template* de operação. No entanto, exceções não se limitam a métodos, não há nenhum mecanismos para tratar o problema de forma eficiente.

CONSTRUÇÃO DO SISTEMA — Booch

APLICAÇÃO

Ciclo de vida	Iterativo
Processamento	*on-line*

EXPRESSÃO

Processo:	Representação:
Modelagem do fluxo das informações	não
Identificação de classes:	
Classes do problema	diagrama de classe
Classes da solução	diagrama de classe
Classes temporais	não
Identificação dos relacionamentos:	
Relacionamento Instância (atributo)	diagrama de classe
Relacionamento classe	não
Relacionamento temporal	não
Relacionamento dinâmico (troca mensagem)	diagrama de objeto
Generalização	diagrama de classe
Agregação	diagrama de classe
Agrupamento	diagrama de classe
Identificação do comportamento	diagrama de objeto
Identificação dos estados de objeto	diagrama de transição de estado
Otimização de classes	sim
Identificação de concorrência	diagrama de tempo
Definição das informações	*templates*
Integração modelo estático e dinâmico	não
Abordagem OO	puramente OO
Estratégia de desenvolvimento	*gestalt*
Gerenciamento da complexidade	comunicação,subsistema e categoria
Adequação ao tratamento de exceções	não

FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO

Clareza de objetivos - não muito claro

Existência de heurísticas - Identificar objetos,operações, relacionamentos e alocar classes em módulos

Exigência de conhecimento - Conhecer conceitos de OO

FACILIDADE DE USO

Existência AA - Sim

Flexibilidade - O método é flexível porque adota um processo iterativo, não sequencial.

Manutenibilidade - Como os módulos de classe são independentes um do outro, uma alteração pode resultar num simples abandono de um módulo de classes não necessitando reprojetar o que já foi feito.

AVALIAÇÃO DO SISTEMA

VERIFICAÇÃO

Não

VALIDAÇÃO

Booch relembra a necessidade de fazer "walkthroughs" do projeto utilizando os diagramas, mas como os diagramas não apresentam notações acessíveis para o usuário, a melhor forma de validar o modelo é utilizar os protótipos.

GERÊNCIA DO SISTEMA

ACOMPANHAMENTO DE CUSTOS E CRONOGRAMA

Não

CONTROLE DE VERSÕES

Não

Figura 4.8 - Tabela de avaliação de Booch

4.2.5. Metodologia "Análise e projeto de sistemas orientada a objetos"

4.2.5.1. Descrição da metodologia

Esta metodologia foi desenvolvida por Baptista em 1992. Ela tem como princípio básico construir o modelo em torno de entidades e relacionamentos, promovendo a reusabilidade, extensibilidade e a modularidade na análise e projeto de sistemas.

A representação desta metodologia é feita através de 7 diagramas e vários *templates* associados. A seguir descreveremos a funcionalidade dos seguintes diagramas:

- **Diagrama de entidades**, nele cada entidade representada contém informações referentes a atributos, operações, restrições, persistência e concorrência. Os tipos de relacionamentos entre entidades são a generalização, a associação entre entidades e a agregação. O diagrama de entidade contém uma parte para interface, em que são colocados os atributos e operações públicas.
- **Diagrama de visões,** onde são definidas as visões das entidades. Uma visão consiste de um subconjunto distinto de informações de uma entidade. Esse diagrama contém o nome da visão, o nome da entidade associada e os subconjuntos de atributos, operações e restrições. O modelo padrão é mostrado na figura 4.9.

Figura 4.9 - Diagrama de visões

- **Diagrama de módulos,** onde entidades e relacionamentos que possuem um significado geral podem ser agrupados em módulos, onde a visibilidade é estabelecida através da definição das entidades públicas. Um diagrama de módulo relaciona as entidades públicas e privadas, as associações e as agregações que compõem um módulo. Os relacionamentos entre os módulos são feitos através das entidades públicas.
- **Diagrama da visão modular**, onde módulos que possuem características diferentes e independentes são vistos de várias formas, denominada visão modular.
- **Diagrama de transição de estados** que representa os eventos, que mudam os estados dos objetos, e as ações ou regras, que são a resposta de um objeto em relação a um evento. Os estados em um diagrama podem ser refinados, possibilitando a definição de diagramas aninhados.
- **Diagrama de comunicação de entidades,** que é utilizado para mostrar a comunicação dos objetos através de mensagens, que podem ser ações, regras e transações.
- **Diagrama de tempo** que documenta o fluxo de controle de uma parte do sistema, onde o fator tempo é crítico.

Nesta metodologia, os passos das fases de análise e projeto são em parte sequenciais e em parte paralelos.

Fase de análise

Na figura 4.10 vemos os passos da fase de análise. Como esses passos são muitos, nos deteremos a mencionar o propósito dessa fase em geral.

A análise identifica a estrutura geral do sistema e atualiza a biblioteca de classes, considerando a generalização (entidades genéricas). Relaciona as entidades selecionadas através de associações, herança e agregação. Identifica as exceções referentes a atributos e operações. Incorpora mecanismos para integração de visões de diversos usuários. Constrói o diagrama de transição de estado e diagrama de comunicação de entidades. Finalmente, faz a especificação formal e validação do modelo.

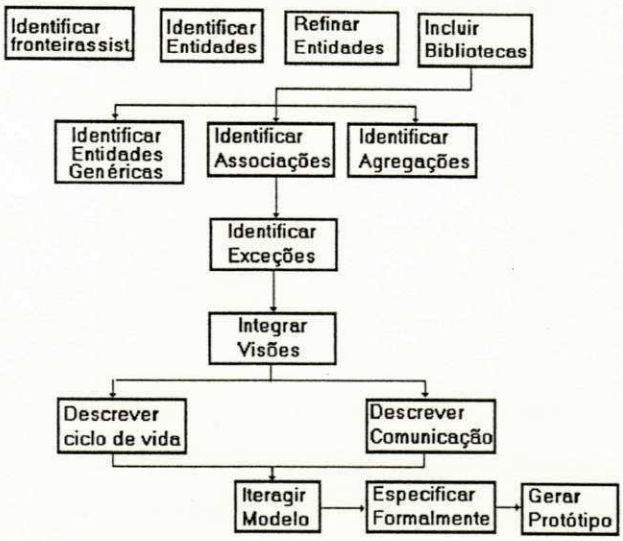

Figura 4.10 - Fases da análise

Fase de projeto

A fase de projeto determina o refinamento do modelo. Ela consiste em definir a estrutura de dados e algoritmos, especificando a arquitetura interna do sistema a partir da especificação obtida na análise. Durante essa fase, algumas decisões de projeto são tomadas, como: definições de módulos, visões de módulos e controle de concorrência e do tempo. Os passos da fase de projeto são mostrados na figura 4.11.

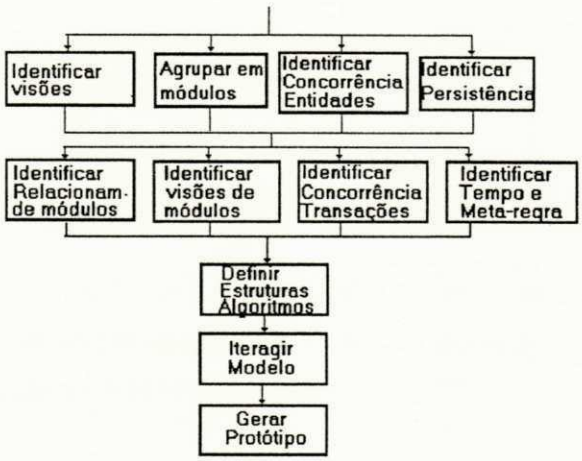

Figura 4.11 - Fases do projeto

4.2.5.2. Avaliação da metodologia

Consideremos a tabela da figura 4.12 contendo a avaliação desta metodologia.

4.2.5.3. Análise crítica

Esta metodologia apresenta mecanismos de comunicação entre os objetos e de modularização. Ela também fornece meios para abordar visões e o tratamento de exceções.

A filosofia do diagrama de transição de estado é inovadora, o diagrama não contém os estados por entidade, mas por operação da entidade. Isto permite acompanharmos somente os estados do objeto, que são afetados por uma determinada operação. Esta é forma de controlar a complexidade comportamental de um objeto.

Neste modelo não há nenhum indício de que exista alternativas para gerenciar a complexidade da estrutura, de forma que o diagrama de entidade tende a ficar muito extenso. Não contém suporte para representar o agrupamento de entidades.

Como a descrição dos passos das fases do processo são muito sucintas, dificultou a nossa análise quanto ao passo de incorporação de mecanismos para integração de visões de diversos usuários. No exemplo dado para a validação da metodologia não existe esse item, de forma que não esclarece que mecanismos são esses, como são integrados esses diagramas de visões?

O modelo utilizado contém vários conceitos para representar o comportamento das classes: operação, ação, regra de comportamento e transação. Esses conceitos deixam o leitor confuso.

Embora a metodologia apresente uma notação expressiva, ela é composta de muitos diagramas (7). O projetista leva tempo até aprender a construí-los.

4.3. Análise comparativa

Da tabela 4.13 podemos inferir que as metodologias de Schiel e Shlaer e Mellor são próprias para ciclos de vida clássicos, enquanto que as demais propõem passos iterativos. Este novo tipo de ciclo de vida propicia uma maior flexibilidade e manutenibilidade do sistema desenvolvido.

A metodologia de Schiel é voltada para sistemas de tempo real devido ao suporte às características temporais.

Quanto ao processo e representação das metodologias, verificamos alguns pontos satisfatórios abordados em todas as metodologias e outros que ainda precisam ser mais reforçados. Os pontos satisfatórios foram: identificação de classes do problema, de atributos, de generalização, dos estados dos objetos e do comportamento, a representação da estrutura estática e a identificação de concorrência. Os pontos fracos foram: identificação de classes da solução, suporte a características temporais, identificação e representação de outros tipos de relacionamentos e abstrações, otimização de classes, representação do comportamento dinâmico (troca de mensagens associadas ao controle de eventos) e integração dos modelos estáticos e dinâmicos.

CONSTRUÇÃO DO SISTEMA	Shlaer	Schiel	Rolland	Booch	Baptista
APLICAÇÃO					
Ciclo de vida	clássico	clássico	iterativo	iterativo	iterativo
Processamento	on-line	TemReal	on-line	on-line	on-line
EXPRESSÃO					
Processo:					
Modelagem do fluxo informações	X	X			
Identificação de classes :					
Classes do problema	X	X	X	X	X
Classes da solução				X	X
Classes temporais		X			
Identificação dos relacionamentos:					
Relacionamento Instância	X	X	X	X	X
Relacionamento classe		X			
Relacionamento temporal		X			
Relacionamento dinâmico		X	X	X	X
Generalização	X	X	X	X	X
Agregação		X	X	X	X
Agrupamento		X	X	X	
Identificação do comportamento	X	X	X	X	X
Identificação dos estados de objeto	X		X	X	X
Otimização de classes			X	X	X
Identificação de concorrência		X	X	X	X
Definicão das informacões	PE,AD e docmtos	tabelas	esquema	templates	templates
Integração modelo estático/dinâmico		X			X
Abordagem OO	combinativ	combinativa	pura	pura	pura
Estratégia de desenvolvimento	top-down	gestalt	top-down	gestalt	NPI
Gerenciamento da complexidade	subsistema	função	comunica-ção	subsistema cat.classe e comun.	estado
Adequação tratamento exceção		X			X
APRENDIZADO					
Clareza de objetivos	X	X	+/-	+/-	+/-
Existência de heurísticas	X	X	X	X	X
Exigência de conhecimento	AE e OO	AE,OO e Petri-Net	OO	OO	OO e Mooz
FACILIDADE DE USO					
Existência de AA	NPI		sim	NPI	Em Imple mentação
Flexibilidade	sequencial	sequencial	paralelos	paralelos	paralelos
Manutenibilidade	fraca	fraca	boa	+/-	+/-
AVALIAÇÃO DO SISTEMA					
VERIFICAÇÃO					Especifica ção formal
VALIDAÇÃO	revisões		regras	protótipos	protótipos

Figura 4.13 - Tabela de análise comparativa

Essas metodologias diferem ao receber as informações dos diversos elementos que elas suportam. A metodologia de Shlaer utiliza linguagem natural para escrever os vários documentos e português estruturado e árvores de decisão para especificar os processos. Schiel utiliza tabelas para descrever os métodos, regras e tempo. Rolland não faz uso de nenhum recurso, escrevendo diretamente na sintaxe do modelo de dados suportado e as metodologias de Booch e de Baptista utilizam *templates*.

Nesta comparação foram identificadas as metodologias de Baptista, Booch e Rolland como puramente orientadas a objetos e as de Schiel e Shlaer como combinativas. Por isto, somente nessas últimas existe modelagem do fluxo das informações em diagrama de fluxo de dados. Monarch [Monarch92] acha que a abordagem combinativa é melhor, porque ela combina o paradigma de orientação a objetos com análise estruturada, embora admita que os conceitos das duas sejam conflitantes. Booch sugere não misturar as duas coisas, porque é difícil construir um sistema orientado a objeto de um modelo que é baseado em decomposição algoritmica.

Quanto ao gerenciamento da complexibilidade podemos dizer que a metodologia de Schiel adota a decomposição funcional tradicional, a de Baptista sugere uma decomposição funcional por objeto, verificando os estados que esse objeto pode assumir quando se processa uma função. A metodologia de Shlaer particiona de forma *top-down* um modelo de informação em vários modelos de informação, e cada modelo é associado a um subsistema, e utiliza a técnica *most-critical-element-first* para gerar os diagramas de estrutura. A metodologia de Booch gerencia a complexidade de 3 maneiras diferentes, dependendo do diagrama modelado: por subsistema (diagrama de subsistema), por categoria de classe (diagrama de categoria de classe) e por comunicação (diagrama de objeto). Na metodologia de Rolland este controle é feito considerando os objetos que se comunicam.

As metodologias de Schiel, Booch e Baptista possuem características que tratam exceções. Mas nas metodologias de Schiel e Booch o suporte é dado somente quanto as restrições dos métodos e na de Baptista é dado também quanto a estrutura das entidades.

De uma forma geral as metodologias são claras quanto ao objetivo e apresentam heurísticas que facilitam o aprendizado do processo da metodologia. As metodologias de Shlaer e Schiel exigem outros conhecimentos do projetistas além dos conceitos de orientação a objetos.

Quanto a avaliação do sistema, as metodologias são fracas na aplicação da técnica de verificação, somente a metodologia de Baptista apresenta uma especificação formal. Com relação a validação, os recursos são os mais variados possíveis, revisões, entrevistas, aplicação de regras e utilização de protótipos.

Das metodologias apresentadas nenhuma delas possuem suporte à gerência do sistema. A existência de apoio automatizado ainda é muito precária.

CAPÍTULO 5

Metodologia FADO

5.1. Introdução

Bancos de dados avançados buscam modelar melhor o mundo real oferecendo uma grande variedade de conceitos (classes, relacionamentos, abstrações, eventos e triggers). A utilização sistemática desses conceitos , oferece facilidades ao projetista para especificar e conceber, progressivamente, o domínio.

Apesar de já existirem metodologias [Booch 91] e [Meyer 88] que tratam da estruturação dos objetos, elas são mais voltadas à programação orientada a objetos do que ao projeto de banco de dados. Nessas metodologias, características como controle de classes e de relacionamentos temporais e controle de eventos não são abordadas. Embora a metodologia de [Rolland 91] forneça uma descrição de ambos os aspectos estáticos e dinâmicos de um sistema de informação, ela é mais voltada ao projeto de eventos do que à especificação de objetos e aos relacionamentos dinâmicos entre eles.

Para cobrir as deficiências acima citadas, apresentamos uma metodologia orientada a objetos chamada FADO. A metodologia FADO inspirou-se nas metodologias [Booch91],[Rolland91] e [Schiel90]. Quanto ao processo da metodologia Booch foi considerada a identificação da semântica das classes e dos mecanismos de operação. Em relação a metodologia de Rolland foi considerada a especificação do comportamento dinâmico dos objetos, através do controle de eventos disparadores de ação e da dependência cronológica entre eventos. Da metodologia de Schiel foram pegas algumas idéias da estrutura estática, objetivando obter um diagrama padrão na área de pesquisas de banco de dados.

Este capítulo está dividido assim: na primeira seção são explicados os componentes gráficos e textuais, na seção seguinte o processo da metodologia FADO. Na terceira seção mostramos a relação entre o processo e a representação FADO, na quarta seção fazemos uma análise crítica da metodologia com relação aos critérios definidos no capítulo 4, e na última seção fazemos uma conclusão.

5.2. Representação FADO

Com o propósito de ilustrarmos a documentação utilizada na representação da metodologia FADO, escolhemos como exemplo um sistema bibliotecário. Parte desse sistema é descrito a seguir:

- Os objetos deste domínio são *Assinante, Obra, Exemplar e GerênciaPedidos*;
- *GerênciaPedidos* é responsável pelo atendimento de pedidos de empréstimo de um exemplar, controle de aquisição, devolução de exemplares, etc;
- Os pedidos de empréstimos dependem da disponibilidade do exemplar e do limite de crédito do assinante para serem atendidos;
- Quando todos os exemplares estão emprestados, e o assinante ainda tem crédito para aquisição, é feita a reserva do exemplar.

Neste trabalho, consideramos as seguintes definições equivalentes: "um objeto da classe A" significa "um objeto instância da classe A" e "estrutura e comportamento da classe A" significa "estrutura e comportamento dos objetos de A".

A metodologia FADO possui a seguinte documentação gráfica e textual, que é representada por tabelas, diagramas, formulários e listas.

5.2.1. Informações contidas nas Tabelas

5.2.1.1. Tabela de ativação entre objetos (TAO)

É uma lista de todas as relações dinâmicas existentes entre os objetos, onde definimos o ciclo de vida de cada objeto, desde sua criação até a destruição, incluindo suas características comportamentais: evento e ação. O comportamento de um objeto é a maneira como ele age ou reage diante de mudança de estado e troca de mensagens.

Nas informações a serem preenchidas, há o conceito de objeto ativo e passivo. Objeto ativo é aquele

Objeto Ativo	Objeto Passivo	Método Chamado	Evento[s]
Gerência Ped	Exemplar	AlteraStEx ConsultaStEx	Atendimento, Infração e Reserva Infração
Gerência Ped	Obra	Consulta Ex.Disp.	Pedido Empr.
Gerência Ped	Assinante	AlteraStAss Aumenta No.empr Diminui No.empr Consulta Ass.Disp.	Atendimento, Infração Atendimento. Reserva Infração Pedido Empr, Infração
Operador Interface	Gerência Ped	Cria Pedido	Pedido Empr
Clock	GerênciaPed	JulgaInfração	TodoDia

Figura 5.1 - Tabela de Ativação entre Objetos

responsável pela invocação de um método pertencente a outro objeto. Objeto passivo é aquele que exibe um comportamento ao ser ativado por outro objeto. Método é o serviço que o objeto passivo pode oferecer aos objetos que tenham visibilidade da sua interface. O evento é o causador do objeto ativo invocar um método do objeto passivo. Consideremos a tabela da dinâmica de objetos da figura 5.1 associada ao exemplo modelado.

5.2.1.2.Tabela de eventos (TEV)

Esta tabela contém especificações dos eventos como: tipo do evento, código do evento , descrição e condição de ocorrência, podendo ser vista na figura 5.2.

Tipo	Código	Nome Evento	Condição de ocorrência do Evento
Externo	E 1	Pedido Empréstimo	Quando há o pedido de empréstimo por um assinante
Interno	E 2	Atendimento	O pedido é atendido se houve um pedido de empréstimo, se Qtd. livros emprestados é menor que a qtd.permitida, e se o exemplar está disponível
Interno	E 3	Reserva Exemplar	O pedido é atendido se houve um pedido de empréstimo, se Qtd. livros emprestados é menor que a qtd.permitida, mas não há nenhum exemplar disponível
Interno	E 4	Infração	O exemplar está reservado há um mes O Exemplar está emprestado há mais de 3 meses
Temporal	E 5	TodoAno	Anualmente há um sorteio de um assinante

Figura 5.2 - Tabela de eventos

5.2.2. Informações dos diagramas

5.2.2.1. Diagrama estático (DE)

Este diagrama é o instrumento utilizado para representarmos o esquema estático, projetando as estruturas das classes e os relacionamentos entre elas. Os tipos de relacionamentos representados são: de instância, de classe e abstrações. A figura 5.3 mostra a estrutura da classe *Assinante*, contendo um relacionamento de instância e as formas de abstrações de generalização e agregação.

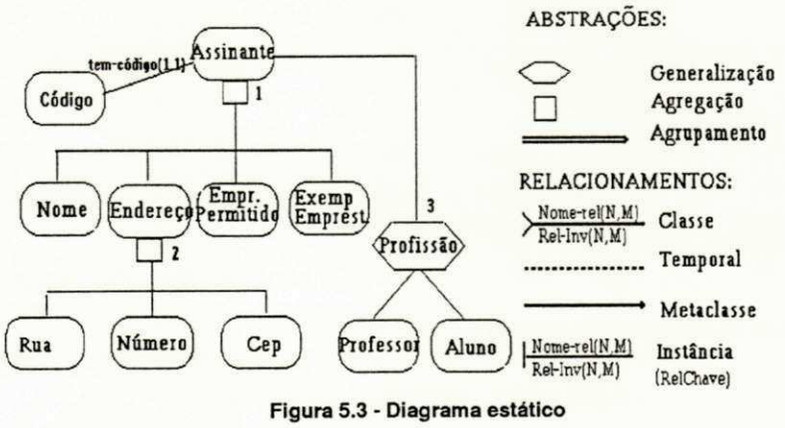

Figura 5.3 - Diagrama estático

5.2.2.2. Diagrama contextual (DCT)

O diagrama contextual fornece ao projetista uma visão geral da aplicação, modelando o comportamento dinâmico do sistema como um todo. Nele, as classes são representadas num nível de abstração mais alto, se comunicando através de mensagens (relacionamento dinâmico). Essas classes são, geralmente, as classes mais superiores de cada estrutura no diagrama estático, ou classes reutilizáveis. No diagrama contextual da figura 5.4, para que o objeto *GerênciaPedidos* envie a mensagem *AltereStEx* para o objeto *Exemplar*, a parte da implementação da classe *GerênciaPedidos* deve ver a interface de *Exemplar*, pois *AltereStEx* é definida em *Exemplar*.

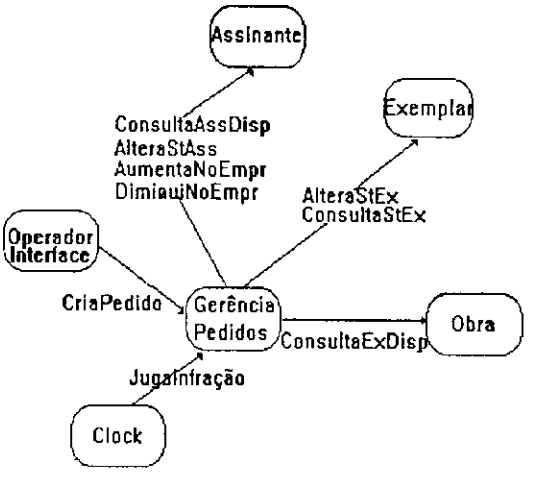

Figura 5.4 - Diagrama contextual

5.2.2.3. Diagrama dinâmico (DD)

Este é o diagrama que representa o comportamento individual de cada classe abordada no diagrama contextual, em função dos eventos associados a ela, disparando ações ou falha-ações do *trigger*, das classes que ela se comunica, e das informações necessárias ao processamento de um determinado método.

A figura 5.5 mostra a simbologia utilizada neste diagrama. A representação dos tipos de eventos é mostrada na figura 5.5 (a). A ligação do evento ao método é representada por fluxo de objeto [figura 5.5 (c)]. Nos fluxos de objetos são passados parâmetros, atributos de uma classe, necessários ao processamento do método ativado.

Os eventos podem aparecer representados de três formas diferentes:

- Gerados por um método [figura 5.5(b)];
- Ativando um método [figura 5.5(c)];

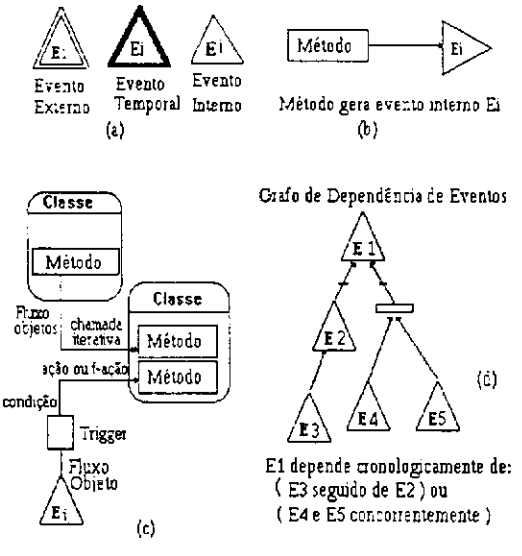

Figura 5.5 - Simbologia do diagrama dinâmico

- Dependentes de outros eventos [figura 5.5(d)][Rolland 91]. Este grafo de dependência entre eventos mostra a relação cronológica de ocorrência de eventos. Eles podem ocorrer em sequência (relacionamento lógico "e"), serem mutuamente exclusivos (relacionamento lógico "ou") ou serem concorrentes [Falquet 88]. O retângulo simboliza a ocorrência dos eventos E4 e E5 concorrentemente.

O diagrama dinâmico de *Exemplar* (objeto em destaque da figura 5.6) mostra duas formas de invocarmos um método de Exemplar:

- Através de um evento, por exemplo, o evento interno *Atendimento* gerado pelo método *CriaPedido* do objeto *GerênciaPedidos* dispara a falha-ação (F1) do *trigger* T1;
- Através da chamada direta do método, por ex: *JulgaInfração* de *GerênciaPedidos* chama *ConsultaStEx* de *Exemplar*, como uma chamada de subrotina.

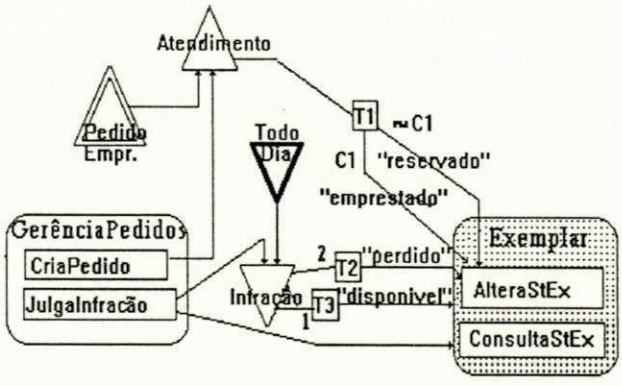

Figura 5.6 - Diagrama dinâmico

5.2.2.4. Diagrama de transição de estado (DTE)

O diagrama de transição de estado representa a evolução dos estados de um objeto e os eventos que causam a transição de um estado para outro.

Podemos seguir a evolução de um objeto no diagrama de transição de estado da seguinte maneira: a partir de um determinado estado, ocorre um evento que causa uma transição para algum outro estado. Na entrada para o estado, os métodos associados aos eventos são executados. A instância está agora num novo estado, e nada acontece com ela até ocorrer outro evento.

Um evento que sai de um estado depende dos eventos que chegam a este estado. Neste diagrama não é determinado se esta dependência é exclusiva ou não, a ordem dos eventos, nem a interação com eventos que acontecem a outros objetos. Tais detalhes são modelados no grafo de dependência no diagrama dinâmico.

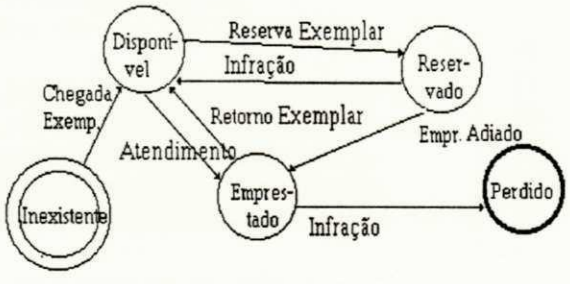

Figura 5.7 - Diagrama de transição de estado

As mudanças de estado de um objeto alteram o valor de um atributo e/ou movem o objeto para outra classe. Este segundo caso é representado pelos relacionamentos pré-pós no modelo TOM. Consideremos o diagrama de transição de estado da classe *Exemplar*na figura 5.7

5.2.3. Informações contidas nos formulários

As informações para definição e implementação das classes do modelo de dados TOM são obtidas através dos formulários. Os formulários existentes são: classe, método, relacionamento, abstração, evento, *trigger*, regra e condição. Existem quatro tipos de formulários de abstração a serem preenchidos: generalização, agregação, agrupamento e herança. Com o objetivo de facilitar a compreensão das informações contidas nos formulários, adotamos as seguintes notações: a vírgula significa "e", uma barra vertical significa "ou", duas barras verticais significa "e/ou".

5.2.3.1. Formulário de classe (FCL)

Cada classe no diagrama estático possui um formulário no formato descrito abaixo. Os dois primeiros campos correspondem ao nome e descrição do significado da classe. Se a classe herdar características semânticas de uma classe existente na biblioteca de metaclasses, essa metaclasse é indicada no campo denominado *especialização*. Se a metaclasse for *V-Class*o relacionamento *versão de* é preenchido. Os subitens do campo *tipo da classe* são informações específicas a serem preenchidas de acordo com o tipo da classe. A lista dos relacionamentos e operações são também definidas. Se a classe possui um diagrama de transição de estado é colocado o nome do diagrama e informado se os relacionamentos pré-pós devem ser considerados.

Classe	: Nome da Classe
Documentação	: Descrição textual
Especialização	: Nome da metaclasse
Versão de	: Nome da classe que possui as versões
Tipo da classe	
Domínio	: Sim| não
Tipo	: Integer| Real| Caracter| Array| ...
Tamanho	: tamanho reservado para tipo
Temporal	: Sim| não
Tempo	: AA:MM:DD:hh:mm:ss
Metaclasse	: Sim| não
Relacionamentos	
Classe	: Lista dos nomes dos relacionamentos
Instância	: Lista dos nomes dos relacionamentos
Generalização	: Lista dos números das abstrações
Agregação	: Lista dos números das abstrações
Agrupamento	: Lista dos números das abstrações
Métodos	
Classe	: Lista dos nomes dos métodos
Instância	: Lista dos nomes dos métodos
Diag transição estados	

5.2.3.2. Formulário de método (FME)

Todas as ações devem estar definidas segundo o padrão do formulário de métodos. Nesse formulário, devem constar informações sobre o nome do método, sua descrição textual, a classe que o possui, seu tipo, os parâmetros de entrada e de saída, as pré-condições e pós-condições, a serem verificadas antes ou depois da execução do corpo da ação e as operações que constituem a ação.

Método	: Nome do método
Documentação	: Descrição textual
Classe do método	: Nome da classe
Tipo do método	: Classe| Instância
ParâmetrosEntrada	: Lista de parâmetros (nome, tipo)
ParâmetrosSaída	: Lista de parâmetros (nome, tipo)
Pré-condição	: Número da condição
CorpoDaAção	: Operações que constituem a ação
Pós-condição	: Número da condição

5.2.3.3. Formulário de relacionamento (FRL)

Para os relacionamentos de classes e de instâncias são utilizados o mesmo formulário. As informações desse formulário são: o tipo do relacionamento, a classe que possui o relacionamento, a classe destino, a cardinalidade do relacionamento, o relacionamento invertido e se o relacionamento é chave para classe. Se o relacionamento for temporal, perguntamos quantos valores são necessários serem guardados e qual a característica do tempo: histórico, transacional e temporal. O modelo padrão está representado abaixo.

TipoRelacionamento	: Classe| Instância
ClasseOrigem	: Nome da classe
Relacionamento	: Nome do relacionamento
Cardinalidade	: Número mínimo, número máximo
ClasseDestino	: Nome da classe
RelacionamentoInverso	: Nome do relacionamento invertido
CardinalidadeInverso	: Número mínimo, número máximo
ChaveParaClasse	: Sim| não
Relacionamento temporal	
QtdValorAnterior	: Número
CaracterísticaTempo	: Histórico| Transacional| Temporal

5.2.3.4. Formulário de generalização (FGE)

Neste formulário associamos cada subclasse a uma única superclasse através do papel. As características do relacionamento envolvido se referem à existência de subclasses disjuntivas e/ou contidas e ao tipo da generalização.

NoAbstração	: Número da abstração
Superclasse	: Nome da superclasse
Subclasses	: Lista dos nomes das subclasses
Papel	: Nome do papel
Parâmetros	: Disjuntivo| | Completo
TipoGeneralização	: Explícito| PelaCondição (NoCondição)

5.2.3.5. Formulário de agregação (FAGR)

Nas informações deste formulário constam as características do relacionamento de agregação entre a classe componente e as agregadas como: se redefinido e/ou explícito e qual o tipo de agregação.

NoAbstração	: Número da abstração
Agregada	: Nome da classe agregada
Componentes	: Lista dos nomes das classes componentes
Parâmetros	: Redefinido| | Exclusivo
TipoAgregação	: Todos| Explícito| PeloRelacionamento (NomeRelacionamento)

5.2.3.6. Formulário de agrupamento (FAGP)

Na descrição do relacionamento de agrupamento entre a classe grupo e a classe elemento devemos especificar o tipo do agrupamento e os parâmetros.

NoAbstração	: Número da abstração
Grupo	: Nome da classe grupo
Elemento	: Nome da classe dos elementos
Parâmetros	: Completo| | Disjuntivo| | OrdenadoPor (NomeRelacionamento)
TipoAgrupamento	: Todos| Explícito| PelaCondição(NoCondição) | PeloRelacionamento(NomeRelacionamento)

5.2.3.7. Formulário de herança (FHE)

Através deste formulário podemos fazer uma herança seletiva às abstrações, de todos os relacionamentos e métodos associados a classe que contém a abstração. Esse formulário é uma complementação dos formulários de generalização, agregação e agrupamento.

NoAbstração	: Número da abstração
Herança de	
Relacionamento	: Lista de (NomeRelacionamento, Tipo (Direta| Computada PeloMétodo (NomeMétodo) | Nenhuma)
Método	: Lista de (NomeMétodo, Tipo (Direta| ComputadaPeloMétodo(NomeMétodo)| Nenhuma)

5.2.3.8. Formulário de evento (FEV)

Nos campos iniciais deste formulário descrevemos informações do evento quanto ao seu nome, descrição, se ele está ativo, quais as regras em que ele aparece e o tipo do evento. Depois as informações variam conforme o tipo do evento. Para os eventos do tipo externo e interno, as informações são o nome do método que gera o evento e a classe do método. Para os eventos temporais, colocamos quando ele inicia, o atraso e o tipo do evento temporal. Um evento temporal pode ser absoluto, periódico ou relativo. Se for relativo ele depende da ocorrência de outro evento, se for periódico, devemos informar qual a periodicidade.

Evento	: Nome do evento
Documentação	: Descrição textual
Ativo	: Sim| não
Regra	: Lista de regras associadas
TipoEvento	: Externo| Interno| Temporal
EventoNãoTemporal	
Método	: Nome do método
Classe	: Nome da classe do método
EventoTemporal	
Quando	: AA:MM:DD:hh:mm:ss
Atraso	: AA:MM:DD:hh:mm:ss
Tipo	: discreto| contínuo
TipoEvTemporal	: Absoluto| Periódico| Relativo
EvTemporalRelativo	
EventoRelativo	: Nome do evento
EvTemporalPeriódico	
Período	: AA:MM:DD:hh:mm:ss

5.2.3.9. Formulário de trigger (FTG)

Este formulário é preenchido para descrever as características de um *trigger*, de acordo com o modelo padrão abaixo. Os campos iniciais se referem ao seu estado de habilidade, prioridade, em quais casos ele é desabilitado e qual a sequência de execução. Os outros campos especificam a condição e quais as ações a serem tomadas quando a condição for verdadeira e/ou falsa.

Trigger	: Nome do trigger
Habilitado	: Sim| não
Prioridade	: Número
CondDesabilitado	: Número da condição
SeqExecução	: Antes| Igual| Depois
Condição	: Número da condição
Ação	: Lista dos nomes dos métodos
Falha-ação	: Lista dos nomes dos métodos

5.2.4.0. Formulário de regra (FRG)

Este formulário associa os eventos aos *triggers* correspondentes.

Regra	: Nome da regra
Eventos	: Lista dos nomes dos eventos
Triggers	: Lista dos nomes dos triggers

5.2.4.1. Formulário de condição (FCON)

As condições são utilizadas nos formulários de método, generalização, agrupamento e de *trigger*. No formulário a seguir os campos indicam o número da condição, sua descrição e estrutura.

NoCondição	: Número da condição
Documentação	: Descrição textual
Estrutura Sentença	: Predicado que constitue a condição
Tempo	: AA:MM:DD:hh:mm:ss
Falha	: Avise| Cancele| Desfaça
Mensagem	: Texto da mensagem da falha

5.3. Processo FADO

O objetivo básico da metodologia FADO é criar a estrutura e comportamento de classes, através da análise de eventos (controle, sequência e tempo) fazendo uso de um processo iterativo e incremental. A metodologia FADO tem como fundamento a flexibilidade e a iteração, onde os passos a serem seguidos não obedecem a uma sequência rígida, podendo ser paralelos.

A figura 5.8 ilustra as fases da metodologia FADO e os instrumentos associados a cada uma delas.

5.3.1. Análise do domínio da aplicação

Esta fase é centrada na análise de todas as aplicações de um dado domínio e na identificação dos objetos comuns entre elas. Para ter a visão do domínio devemos construir um modelo genérico do domínio e refiná-lo para acomodar as similaridades e diferenças com as aplicações já existentes. Este processo de descrever uma aplicação a partir das já existentes faz uso do paradigma de orientação a objetos: reusabilidade. Neste caso sugerimos construir uma biblioteca de classes.

Os passos desta fase são descritos a seguir.

5.3.1.1. Identificar classes e objetos

Para identificar objetos o projetista deve tentar a familiarização com o vocabulário do domínio do problema, avaliando as seguintes informações: coisas tangíveis, funções desempenhadas por pessoas ou organizações (médico, contribuinte, empregado), especificações relacionadas com estoques ou fabricação (modelo de um eletrodoméstico).

A identificação de objetos não se restringe a esta fase, à medida que avançamos no processo de desenvolvimento e começamos a tratar com aspectos do mundo computacional novos objetos podem surgir, o que reafirma a não linearidade do processo.

A identificação de classes provém de uma análise de semelhanças da estrutura e comportamento dos objetos já identificados.

O produto deste passo é uma lista de classes, preenchendo algumas informações do formulário de classe, e a atualização da biblioteca de classes. Já pode ser traçado um esboço dos diagramas estáticos.

Para obter um ótimo desempenho de reutilização de classes, damos alguns lembretes ao projetista:

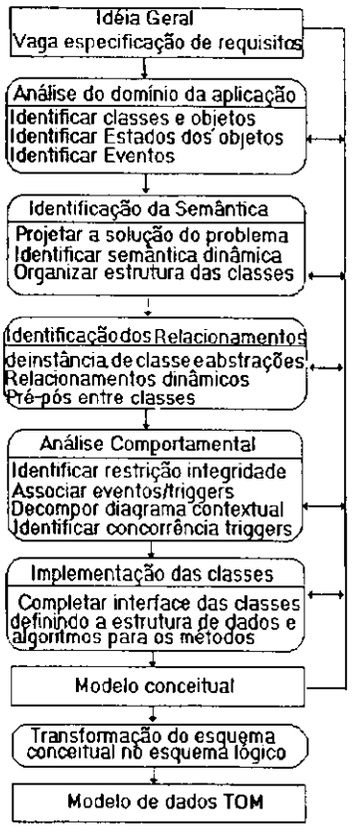

Figura 5.8 - Fases da metodologia

- Armazene métodos e atributos de classe no nível de abstração mais alto. Pois assim, um número maior de subclasses podem compartilhá-los.
- Dê nome idênticos a métodos quando uma classe se comunica com outras classes para realizar operações similares.
- Reduza o tamanho dos métodos. Métodos menores são mais facilmente herdados.
- Elimine os métodos de uma superclasse que são frequentemente sobrescritos em vez de herdados por suas superclasses. Assim, a superclasse se torna mais abstrata e mais útil.
- Em todo o ciclo, as classes são refinadas e reorganizadas, e quando for descoberta que uma classe é genérica o bastante para ser reutilizada, ela deve ser incluída na biblioteca de classes.

5.3.1.2. Identificar estados de objetos

Para qualquer objeto que tenha um ciclo de vida, devemos incluir um atributo *estado* no modelo de informação. [Shlaer 90]. Por exemplo, consideremos a existência de *EstadoDoExemplar* para a classe *Exemplar*. O domínio desta informação é a lista de todos os possíveis estados de um *Exemplar*, definidos na figura 5.7. Num dado momento, o objeto *Exemplar* se encontra *Emprestado*, mas pelas obervações feitas, ele poderá passar para um dos estados: *Disponível, Reservado ou Perdido.*

O produto deste passo é uma relação de estados de objetos.

5.3.1.3. Identificar eventos

A partir dos objetos identificados e de seus possíveis estados, é feita uma análise das ações sobre esses objetos. Essas ações são disparadas diante da ocorrência de eventos.

Para identificar eventos é necessário fazermos um levantamento das informações como: nome dos eventos, descrição dos eventos, as objetos passivos responsáveis pela ação e os objetos ativos que se comunicam com os passivos [Brand 88].

Para analisar a descrição dos eventos devemos verificar a sintaxe da linguagem, que descreve o domínio do problema, onde: *Sujeito* é geralmente o objeto responsável pela resposta; O *objeto* contém os objetos envolvidos; O *verbo* descreve as ações; *Objetos depois de preposições* são objetos ativos. Por exemplo, "Gerência de pedidos recebe uma solicitação de empréstimo de exemplar de um assinante".

Objeto passivo = Gerência de Pedidos;
Objeto ativo = Assinante
Ação = Receber pedidos;
Evento = Entrega solicitação;

Os eventos temporais podem ser modelados de forma análoga aos eventos externos, a única diferença é que agora existe uma indicação de tempo ao invés de um objeto ativo.

O produto deste passo é a construção da tabela de eventos e do diagrama de transição de estado. A tabela de eventos serve para checar a existência de todos objetos encontrados. Isto pode ser feito analisando-se cada evento e identificando qual objeto que detecta ou reage a qual evento. Por vezes descobrimos algum objeto despercebido. Além de servir à consistência das informações, eventos podem ser usados para controlar restrições de integridade. O formulário de eventos pode ir sendo preenchido aos poucos.

5.3.2. Identificação da semântica das classes

Esta fase refere-se a tomadas de decisões de projeto e a obtenção e otimização da semântica das classes, considerando as funções e limitações dos objetos instâncias da classe.

5.3.2.1. Projetar a solução do problema

Esta tarefa é realizada no início do projeto e se refere a tomada de decisão sobre como os objetos colaboram, para fornecermos o comportamento, que satisfaça as necessidades do problema do domínio. Por exemplo, suponha que a chefia da biblioteca queira saber todos os infratores (assinantes que não devolveram os exemplares dentro de 3 meses). Alguns objetos colaboram para fornecer esta informação:

Exemplar, Assinante e GerênciaPedidos. Dentre algumas alternativas possíveis de como eles interagem, o funcionamento escolhido foi: "Todo dia haverá uma procura nos pedidos de empréstimos, controlados pela *GerênciaPedidos*, em seguida será pego o nome dos exemplares e dos assinantes nos respectivos objetos, e depois um relatório será enviado à chefia". Decidir qual a melhor alternativa é uma tarefa difícil, pois envolve fatores como: estimativa de tempo e custo, e confiabilidade da solução proposta.

O produto deste passo é a definição de um funcionamento da solução descrevendo as funções básicas que o sistema terá. Uma vez que o projetista decide qual o funcionamento utilizado, o próximo passo é distribuir o trabalho entre os objetos, por definir métodos nas classes apropriadas.

5.3.2.2. Identificar a semântica dinâmica das classes

Neste passo descobrimos o significado de cada objeto instância da classe, verificando que tipo de informações ele precisa receber e o que tem a oferecer dentro de suas limitações. Para este propósito é construída a tabela de ativação entre objetos. A decisão de como os objetos interagem entre si, isto é, qual é o objeto ativo do objeto passivo será discutida na próxima fase.

5.3.2.3. Organizar a estrutura global das classes

Por várias vezes voltamos aos diagramas estáticos já projetados, para verificar suas semânticas e, se necessário, reorganizar a hierarquia. Reorganizar a hierarquia significa utilizar abstração, *generalizando* o projeto, diante das semelhanças entre classes ou *especializando-o* ao descobrir novas classes. O objetivo de organizar a estrutura é melhorar a semântica das classes.

Geralmente, quando definimos as fronteiras de uma classe, tomamos decisões que mudam o significado de outra classe. Por exemplo, consideremos numa companhia aérea, uma classe cujas instâncias representam *vôos*. O controle de vôos para uma determinada época do ano, como colocar vôos extras ou cancelá-los, fica muito complicado se tiver que ser realizado nessa mesma classe. Então criamos uma classe de *GerênciadeVôos*, implementando esse controle.

Esta mudança no projeto de criar ou fundir classes, redefinindo o significado destas, é de carácter evolutivo, isto é, por várias vezes voltamos a este ponto. Este processo termina quando não há mais nenhuma classe ou métodos a serem descobertos, ou mesmo que haja mais algum, eles possam ser implementados a partir dos componentes de software utilizáveis.

O produto deste passo é a alteração dos diagramas estáticos já existentes e da tabela de ativação entre objetos.

5.3.2.4. Apresentar um protótipo

Tendo encontrado as classes envolvidas e as funções essenciais do sistema, partimos para esboçar uma solução. É importante identificar componentes de software reusáveis, pois fazemos uma análise "bottow-up" de classes já existentes, sendo mais rápido construir um protótipo utilizando classes já prontas e testadas.

5.3.3. Identificação dos relacionamentos das classes

Nesta fase descobrimos como as coisas interagem dentro do sistema, estabelecendo os relacionamentos de instância, de classe, temporal, abstrações, relacionamentos dinâmicos e relacionamentos pré-pós.

5.3.3.1. Identificar relacionamentos e abstrações

Este passo tem como objetivo definir os relacionamentos e abstrações entre as classes, verificando suas características temporais e de versionamento. As informações dos formulários de relacionamento e de abstração devem ser preenchidas e os diagramas estáticos refinados.

5.3.3.2. Identificar relacionamentos dinâmicos

Neste passo determinamos os relacionamentos dinâmicos entre os objetos através de mensagens. Na verdade, a identificação do que se pede, já foi realizada em uma grande parte, no momento em que a semântica das classes foram analisadas. Por isto agora se analisa, principalmente, o destino das mensagens dos objetos. O que não significa que, neste momento, não se possa descobrir novos métodos, coisa aliás muito corriqueira (lembrar sempre que os passos não são sequenciais).

A tabela de ativação entre objetos deve ser refinada para associar objetos ativos a objetos passivos. Após esta associação é importante notarmos aqueles objetos que são ativos e nunca são passivos e não considerá-los mais objetos na modelagem. Esses objetos se situam fora do escopo da aplicação e não oferecem serviços (métodos) a outros objetos.

O produto deste passo é a construção do diagrama contextual, representando os relacionamentos dinâmicos entre os objetos, descritos a nível das respectivas classes. A tabela de ativação entre objetos dá apoio à construção desse diagrama, por exemplo, para o diagrama contextual da figura 5.4, verificar a tabela da figura 5.1. As informações de um método podem ser colocadas no formulário de métodos.

5.3.3.3. Identificar relacionamentos pré-pós

A partir do diagrama de transição de estados construído, obtemos os relacionamentos pré-pós, que acompanha a evolução de um objeto. Neste passo devemos atualizar o formulário de classe, dizendo se

esses relacionamentos são considerados no diagrama de transição de estados.

5.3.4. Análise comportamental do objeto

Nesta fase nós identificamos as restrições de integridade necessárias para manter o banco de dado íntegro, depois, fazemos a modelagem do mecanismo de regras ativas do TOM.

5.3.4.1. Identificar restrições

Existem três tipos de restrições a serem identificadas:

- Condição de ocorrência de evento - define quando um evento acontecerá;
- Condição de *trigger* - determina a ocorrência ou não da ação disparada por um evento;
- Condição de método - são condições a serem satisfeitas antes e depois da execução de um método.

Para descobrir as condições de *triggers* e métodos, aconselhamos retirá-las das condições de ocorrência dos eventos. Por exemplo, na tabela de eventos da figura 5.2, as condições para ocorrência do evento *Atendimento* são que:

- Deva haver primeiro um *PedidoEmpréstimo*;
- A quantidade de empréstimo permitida deva ser menor que a quantidade de exemplares já pedidos;
- Existam exemplares disponíveis.

As duas últimas condições passarão a ser condições para que as ações de um *trigger* aconteçam, no caso T1. E a primeira condição mostra uma dependência cronológia de *Atendimento* por *PedidoEmpréstimo*, representada no diagrama dinâmico da figura 5.6. Avaliando, também, as condições de ocorrência do evento *ReservaExemplar* percebemos que se não existirem exemplares disponíveis o assinante não será atendido, havendo reserva de um exemplar para ele. Portanto, as ações disparadas pelo evento *ReservaExemplar* passarão a ser falha-ações do *trigger* T1.

Neste passo fazemos o refinamento da tabela de evento e do formulário de método ao colocarmos as condições, e a construção do formulário de *trigger*, associando a condição do *trigger* às ações do *trigger*. Mostramos abaixo o formulário preenchido para o *trigger* T1.

Trigger	: T1
Habilitado	: Sim
Condição	: C1
Ações	: AlteraStEx("Emprestado"), AlteraStAss ("Atendido"), AumentaNoEmpr
Falha-ações	: AlteraStEx("Reservado"), AlteraStAss ("NãoAtendido"), AumentaNoEmpr

No formulário do *trigger* T1, aparece o número da condição, cuja estrutura deve ser especificada ao preencher o formulário de condição abaixo:

NoCondição	: C1
Documentação	: Condição para a ocorrência das ações de T1
Estrutura Sentença	: StatusExemplar = "Disponível" AND QtdEmprestada < QtdPermitida
Falha	: Avise
Mensagem	: "Assinante não atendido"

5.3.4.2. Associar eventos a triggers

É uma forma de determinar as ações, a serem executadas em consequência da ocorrência dos eventos, que comporão uma regra. O produto deste passo é o preenchimento do formulário de regra considerando as informações contidas na tabela de ativação entre objetos e nos formulários de *triggers*. A seguir, mostramos dois formulários de regras R1 e R2 preenchidos para os eventos *Atendimento* (E2) e *Infração* (E4), respectivamente.

Regra	: R1
Eventos	: E2
Triggers	: T1

Regra	: R2
Eventos	: E4
Triggers	: T2,T3

5.3.4.3. Representar o comportamento dinâmico

A partir da decomposição de cada classe contida no diagrama contextual e das informações das tabelas, construímos um esquema dinâmico por classe, chamado de diagrama dinâmico.

É necessário fazer a integração do esquema estático e dinâmico para cada classe. Após essa integração o diagrama dinâmico representará os relacionamentos de instância, de classe, abstrações e métodos encapsulados no mesmo símbolo representativo da classe.

Considerando esta relação entre o esquema estático e dinâmico, há, geralmente, semelhanças entre a estrutura das classes e estrutura das ações [Rolland 89]. Por exemplo, consideremos uma ação sobre a classe *Obra* , que é um agrupamento de *Exemplar*[figura 5.9 (a)]. Ela pode ser de duas maneiras diferentes:

- Simples : cadastramento de um novo *Exemplar a Obra*.
- Iterativa: A ação de modificar a tabela de preço é a execução iterativa de modificar o preço de todas instâncias de *Exemplar*[figura 5.9(b)].

O projetista deve usar esta semelhança para melhorar a representação do comportamento dos objetos de acordo com a estrutura das classes.

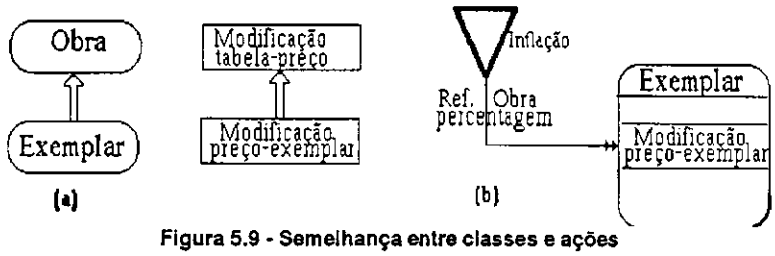

Figura 5.9 - Semelhança entre classes e ações

5.3.4.4. Identificar concorrência entre os métodos

Para identificar a concorrência entre os métodos, em relação a um problema específico, devemos verificar os eventos que atuam no sistema. E depois considerar a concorrência semântica dos objetos que reagem a esses eventos.

O produto deste passo é a complementação do formulário de *triggers* definindo as ações para executarem antes, igual ou depois ao método que gerou o evento do *trigger* e associar a cada *trigger* uma prioridade. Por exemplo, no formulário da regra R2, vemos que o evento *Infração* aciona dois triggers :T2 e T3. O primeiro a ser avaliado é o T2 por ter prioridade 2, e o segundo é o T3 com prioridade 1. Se as condições do T2 forem satisfeitas, então as ações *AlteraStExe AlteraStAss*são ativadas, antes de terminar a execução do método *JulgaInfração*, que ativou o evento.

Trigger	: T2
Habilitado	: Sim
Prioridade	: 2
SeqExecução	: Antes
Condição	: C2
Ações	: AlteraStEx("Perdido"), AlteraStAss ("Removido")

5.3.4.5. Apresentar um protótipo

Neste momento é importante validar o projeto, apresentando novamente um protótipo. Durante esta inspeção, devemos verificar se as normas estabelecidas para o modelo conceitual foram respeitadas, se todos os elementos referenciados foram completamente descritos e, principalmente, se o sistema atende às necessidades do usuário.

5.3.5. Implementação de classes de objetos

Este é o ponto em que temos a visão de dentro de cada classe, para decidir como ela será implementada. Nesta fase é definida a estrutura dos dados e algoritmos para as operações. O produto deste passo é o refinamento da estrutura de classe e a complementação de cada formulário de classe e de método.

5.3.6. Transformação do esquema conceitual no esquema lógico

A partir das informações contidas nos formulários somos capazes de obter o esquema lógico do sistema, de acordo com a sintaxe do TOM definida no capítulo 3. No ambiente DYNAMO, esse esquema lógico deverá ser transformado num esquema interno, para que possa ser utilizado pelo gerenciador de objeto para monitorar os acessos aos objetos.

5.4. Processo FADO X Produto FADO

Uma característica de uma modelagem orientada a objetos é que não há uma fronteira bem definida entre análise e projeto. É difícil determinar onde a análise termina e onde começa o projeto. Na análise orientada a objetos, o projetista se preocupa em descobrir os objetos que têm significado (semântica) no problema da aplicação, modelando suas estruturas. Enquanto que no projeto orientado a objetos, o projetista se preocupa com os objetos que são parte da solução do problema, modelando seus comportamentos dinâmicos. [Monarchl 92].

Na figura 5.10 vemos a relação entre análise e projeto. A representação do domínio do problema (visto na análise) e do domínio da solução (vista no projeto) são menores do que os problemas do mundo real. O domínio da solução inclui todo ou parte do domínio do problema e mais algumas características próprias da solução

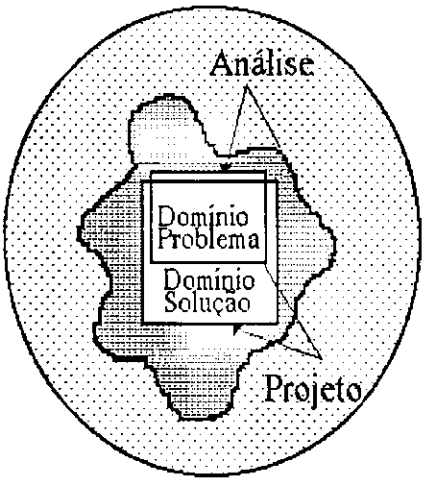

Figura 5.10 - Relacionamento entre análise e projeto

Na figura 5.11 traçamos a alternativa mais adequada para acompanhar os passos do processo FADO e verificar o produto obtido em cada passo. Este roteiro não é rigoroso, como já dissemos os passos não obedecem a uma sequência rígida e alguns podem ser opcionais.

Os passos 1 a 4 são sequenciais. Uma vez proposto o funcionamento da solução podemos voltar ao passo 1 (para encontrar classes e eventos da solução) e/ou identificar a semântica das classes já encontradas e/ou identificar as restrições dos eventos. Um protótipo pode ser construído se o projetista quiser validar a solução encontrada.

Os passos 8 e 10 podem ser paralelos. O passo 6 depende da aplicação da generalização, e tal ação

pode resultar na criação ou eliminação de classes, sendo necessário voltar ao ponto "A" para criar classes e/ou redefinir a semântica das classes .

Para que o passo 9 começe é necessário ter sido identificado as formas de abstração e os relacionamentos entre classes, mas não precisa que os passos 8d , 10 e 11 já tenham sido finalizados. Após identificar as condições dos eventos podemos identificar as restrições tanto dos métodos como dos triggers. A identificação dessas restrições são bastante iterativas, pois as condições devem dar total segurança da integridade do banco de dados. Após ter descoberto os triggers, já podemos verificar a concorrência entre eles e associá-los aos eventos.

O próximo passo é a representação do comportamento dinâmico com a construção dos diagramas dinâmicos. A construção de um protótipo é útil para validar todo o projeto. No passo 16 o projetista termina

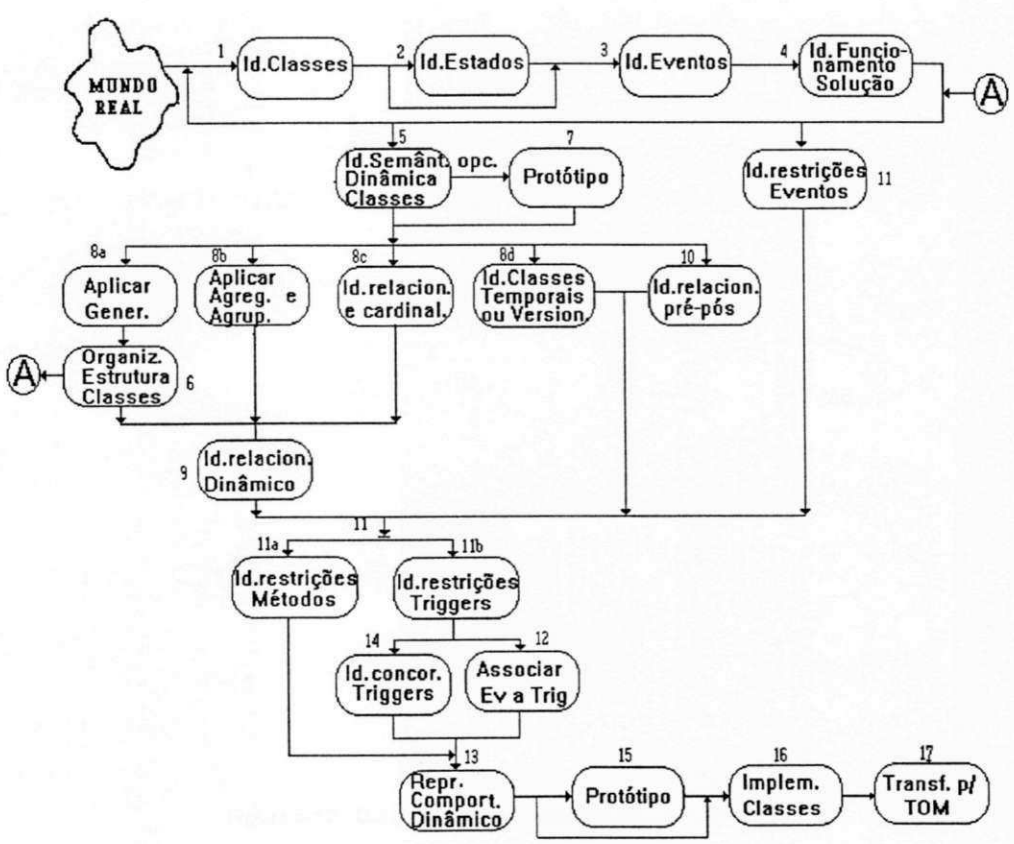

Figura 5.11 - Sequencialidade do processo FADO

de especificar o sistema com a definição de algoritmos para implementação. O último passo é necessariamente a transformação do esquema conceitual nas estruturas da linguagem TOM.

A figura 5.12 mostra os passos do proceso (quadrado) e que recursos eles manuseiam (círculo). A descrição dos processos que corresponde a numeração pode ser vista na figura 5.11, e a descrição dos recursos que corresponde a abreviação é mostrada no item 5.2.

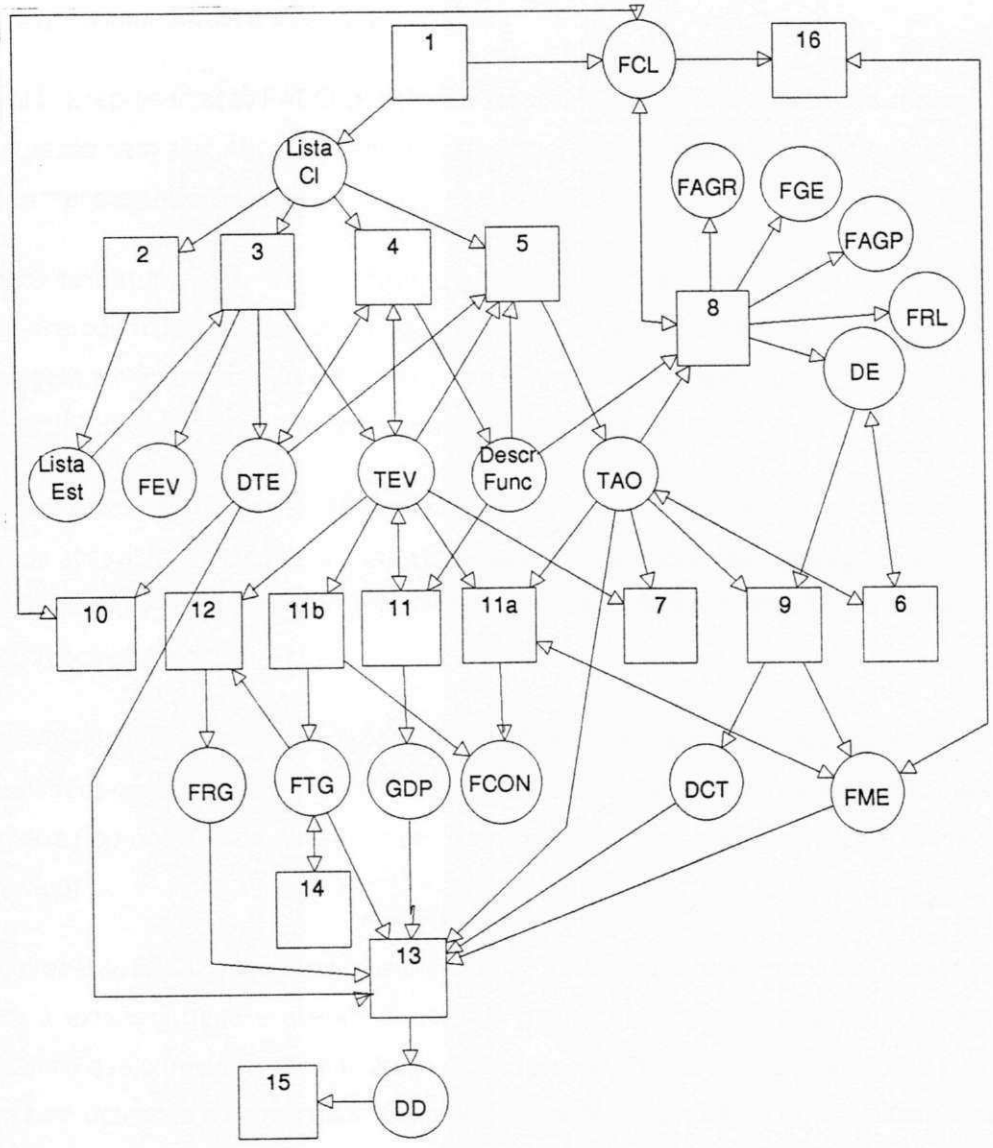

Figura 5.12 - Diagrama de processos e recursos utilizados

5.5 Análise crítica

A tabela da figura 5.13 mostra a avaliação da metodologia FADO, segundo os critérios definidos no

capítulo anterior. A metodologia FADO surgiu para atender os projetistas que desejam obter o esquema lógico do TOM, alcançando com profundidade os conceitos de orientação a objetos e o mecanismo de regras ativas.

Quanto ao processo FADO, o modelo proposto permite flexibilidade no acompanhamento dos passos do processo. Os passos são definidos de maneira completa e detalhada, especificando o que fazer, como fazer (sugere heurísticas) e qual o produto final.

Quanto a representação FADO, podemos dizer que existe uma quantidade boa de diagramas (4), com uma notação bem singular, por exemplo, não existem várias maneiras de representar um objeto ou a troca de mensagens entre eles.

FADO define um diagrama dinâmico para representar a troca de mensagens entre objetos e os eventos que os interagem. Nesse diagrama, podemos acompanhar a dependência dos eventos, a concorrência dos triggers e o tratamento de exceções pelas falha-ações, evitando construir um diagrama de tempo [Booch91][Baptista92] para acompanhar a execução dos métodos.

Constatamos não haver necessidade de definir diagramas de módulos por dois motivos: devido ao poder de abstração do diagrama contextual, obtemos a compreensão do problema como um todo. O segundo motivo é que as classes no TOM, são como as de Smaltalk, elas servem como unidade básica de decomposição modular.

A estratégia utilizada durante o processo FADO é a mistura das técnicas *top-down* e *bottow-up*. Há necessidades concorrentes de analisar o sistema partindo de num nível de abstração mais alto para um mais baixo (*top-down*) e, de implementá-lo fazendo uso de classes já definidas das bibliotecas de classes (*bottow-up*).

No processo FADO é permitido reusabilidade, prototipagem e iteração entre os passos para acomodar melhor à extensibilidade e alterabilidade. Dos modelos de ciclo de vida considerados (capítulo 2) verificamos que o modelo cascata não se adapta a esse processo, porque ele exige que as atividades sejam bem definidas e sequenciais. O modelo baseado em protótipos possui capacidades funcionais limitadas, baixa confiabilidade e desempenho ineficiente. O modelo baseado em reutilização não é adequado, pois ele depende muito de técnicas e ferramentas para criar, armazenar, recuperar os componentes de softwares reusáveis. E no momento ainda são poucos os ambientes de programação orientados a objetos que possuem suporte à biblioteca de classes. O processo FADO se adapta ao modelo de iteração dada sua flexibilidade e possibilidade de reusabilidade das classes

A metodologia descrita é baseada nas seguintes características:

- Modelagem da estrutura e relacionamentos das classes;

CONSTRUÇÃO DO SISTEMA — FADO

APLICAÇÃO

Ciclo de vida	Iterativo
Processamento	Tempo real

EXPRESSÃO

Processo da metodologia:	Representação:
Modelagem do fluxo das informações	
Identificação de classes :	
Classes do problema	DE, DD, DCT
Classes da solução	DE, DD, DCT
Classes temporais	DE
Identificação dos relacionamentos;	
Relacionamento Instância (atributo)	DE
Relacionamento classe	DE
Relacionamento temporal	DE
Relacionamento dinâmico (troca mensagem)	DCT, DD
Generalização	DE
Agregação	DE
Agrupamento	DE
Identificação do comportamento	DCT, DD
Identificação dos estados de objeto	DTE
Otimização de classes	sim
Identificação de concorrência	DD
Definição das informações	Formulários e Tabelas
Integração modelo estático e dinâmico	DD
Abordagem OO	Pura
Estratégia de desenvolvimento	*gestalt*
Gerenciamento da complexidade	Por assunto
Adequação ao tratamento de exceções	sim

FACILIDADE DE APRENDIZADO PARA CONSTRUÇÃO

Clareza de objetivos	sim
Existência de heurísticas	sim
Documentação	[Furtado93]
Exigência de conhecimento	Conhecer conceitos de OO

FACILIDADE DE USO

Existência de AA - Ferramenta definida, mas ainda não implementada

Flexibilidade - Os passos são iterativos, permitindo a reexecução das fases anteriores sem comprometer o que já foi feito.

Manutenibilidade - As alterações são facilitadas pelo uso de biblioteca de classes, uso de protótipos e independência de dados.

AVALIAÇÃO DO SISTEMA

VERIFICAÇÃO

não

VALIDAÇÃO

O protótipo facilita o entendimento e a validação do sistema, pois pode ser utilizado para simular os diálogos e as funções do sistema. A representação FADO possui uma linguagem gráfica e textual com notação acessível aos usuários.

GERÊNCIA DO SISTEMA

ACOMPANHAMENTO DE CUSTOS E CRONOGRAMA

Não

CONTROLE DE VERSÕES

Sim. O modelo TOM suporta controle de versões, e a representação delas em FADO pode ser feita no formulário de classe e no diagrama estático.

Figura 5.13 - Tabela de avaliação da FADO

- Modelagem do comportamento dinâmico dos objetos em função dos eventos;
- Modelagem da semântica dinâmica de objetos e métodos, acompanhando a evolução dos objetos e a ordem de execução dos métodos, respectivamente.

Existem heurísticas que servem como normas de qualidade para auxiliar na construção da representação, dando suporte ao processo de verificação do produto.

A metodologia FADO fornece suporte às características temporias, nos seguintes momentos:

- Classes Temporais - durante a descrição da classe no formulário de classe é especificado durante quanto tempo são guardadas as intâncias e no diagrama estático ela é representada diferentemente das outras.
- Evento Temporal - durante o preenchimento do formulário de eventos, devemos dar informações específicas desse evento, verificando a existência de eventos periódicos e relativos e no diagrama dinâmico ele é representado diferentemente dos outros.
- Relacionamento Temporal pré-pós - esses relacionamentos são representados no diagrama de transição de estados.
- Relacionamento com valores anteriores - durante a descrição do relacionamento de instância no formulário de relacionamentos é especificado quantos valores devem ser guardados.
- Condição Temporais - durante a descrição da condição no formulário de condição é especificado o tempo no qual o predicado era verdadeiro.

Na tabela comparativa da figura 5.14, são colocadas as características de todas as metodologias estudadas. As cinco primeiras metodologias já foram comentadas no capítulo 4 item 4.3 e a metodologia FADO no item 5.5. De uma forma geral nós queremos mostrar nesta figura o porquê da existência de uma nova metodologia no ambiente DYNAMO, se já existia a metodologia de Schiel (POKER).

Embora as metodologias FADO e POKER objetivem gerar o esquema de dados TOM, elas apresentam algumas diferenças básicas:

- Ambas são voltadas para sistemas de tempo real devido ao suporte às características temporais, mas somente FADO explorou modelar os eventos temporais, os relacionamentos pré-pós nos diagramas de transição de estados e as condições temporais.
- A metodologia FADO aborda a decomposição do problema em torno dos objetos, desde a fase inicial do desenvolvimento, enquanto que POKER começa com decomposição funcional e depois provém heurísticas para convertar os diagramas de fluxos de dados no modelo de objetos.
- O processo FADO é muito diferente do processo POKER, sendo os passos iterativos e não obedecendo a uma sequência como no caso do POKER, sendo melhor a manutenibilidade do produto em desenvolvimento.
- A FADO oferece suporte a validação do produto através de regras de completeza (definidas na ferramenta FADO, assunto do capítulo 7), e da utilização de protótipos.
- Permite a reutilização de classes sugerindo a criação de uma biblioteca de classes.
- Para utilizar a metodologia FADO não é necessário ter conhecimentos de análise estruturada e das rede de Petri.

CONSTRUÇÃO DO SISTEMA	Shlaer	Schiel	Rolland	Booch	Baptista	FADO
APLICAÇÃO						
Ciclo de vida	clássico	clássico	iterativo	iterativo	iterativo	iterativo
Processamento	on-line	TemReal	on-line	on-line	on-line	Tempo Real
EXPRESSÃO						
Processo:						
Modelagem do fluxo informações	X	X				
Identificação de classes :						
Classes do problema	X	X	X	X	X	X
Classes da solução				X	X	X
Classes temporais		X				X
Identificacão dos relacionamentos:						
Relacionamento Instância	X	X	X	X	X	X
Relacionamento classe		X				X
Relacionamento temporal		X				X
Relacionamento dinâmico		X	X	X	X	X
Generalização	X	X	X	X	X	X
Agregação		X	X	X	X	X
Agrupamento		X	X	X		X
Identificação do comportamento	X	X	X	X	X	X
Identificação dos estados de objeto	X		X	X	X	X
Otimização de classes			X	X	X	X
Identificação de concorrência		X	X	X	X	X
Definicão das informacões	PE,AD e docmtos	tabelas	esquema	templates	templates	Formulá-rio,tabela
Integração modelo estático/dinâmico		X			X	X
Abordagem OO	combinativ	combinativa	pura	pura	pura	pura
Estratégia de desenvolvimento	top-down	gestalt	top-down	gestalt	NPI	gestalt
Gerenciamento da complexidade	subsistema	função	comunica-ção	subsistema cat.classe e comun.	estado	assunto
Adequação tratamento exceção		X			X	X
APRENDIZADO						
Clareza de objetivos	X	X	+/-	+/-	+/-	
Existência de heurísticas	X	X	X	X	X	X
Exigência de conhecimento	AE e OO	AE,OO e Petri-Net	OO	OO	OO e Mooz	OO
FACILIDADE DE USO						
Existência de AA	NPI		sim	NPI	Em imple-mentação	Só há o projetoda interface
Flexibilidade	sequencial	sequencial	paralelos	paralelos	paralelos	paralelos
Manutenibilidade	fraca	fraca	boa	+/-	+/-	+/-
AVALIAÇÃO DO SISTEMA						
VERIFICAÇÃO					Especifica çãoformal	
VALIDAÇÃO	revisões		regras	protótipos	protótipos	regras protótipos
GERÊNCIA DO SISTEMA						
CUSTOS E CRONOGRAMA						
CONTROLE DE VERSÕES						X

Figura 5.14 - Tabela comparativa geral

CAPÍTULO 6

Estudo de caso

O desenvolvimento de um ambiente de aquisição automática de conhecimento

6.1. O domínio de aquisição automática de conhecimento

O domínio que trabalhamos foi o da aquisição automática de conhecimento para sistemas baseados em conhecimentos. Esse domínio foi assunto de uma tese de mestrado do Departamento de Sistemas e Computação da UFPb, na área de Inteligência Artificial, originando o ambiente A4 [Vasco93]. Os motivos pelos quais o escolhemos foram a proximidade com o usuário, facilitando esclarecimentos dos requisitos e interação, e a abrangência do domínio, possibilitando explorar exemplos mais completos. Além disso, em função do domínio assemelhar-se a um CASE para sistemas baseados em conhecimentos, pudemos explorar características de tempo e versionamento próprias da metodologia FADO.

A tarefa de aquisição de conhecimento (AC) automática objetiva diminuir as dificuldades encontradas no processo manual de AC, no qual há um engenheiro de conhecimento que, através de interações com o especialista, é responsável por montar uma base de conhecimento sobre o domínio. Os métodos de AC automática buscam minimizar a presença do engenheiro do conhecimento e até mesmo do especialista nesse processo.

Dentre os vários métodos de AC automática existentes, os que se baseiam em indução a partir de exemplos são os mais explorados. Esses métodos buscam, a partir de exemplos reais, formar uma base de conhecimento em alguma forma de representação de conhecimento. Essa base de conhecimento é obtida através da aplicação de algoritmos estatísticos baseados em indução nos exemplos de um domínio.

Figura 6.1 - Processo de AC indutiva

A figura 6.1 resume a atividade de AC indutiva a partir de exemplos.

Os exemplos, que servem como entrada à geração de conhecimento, são representados por um conjunto de pares atributo/valor, que concluem um certo elemento classificador. A figura 6.2 mostra um conjunto de exemplos, conhecido como tabela de exemplos, no qual podemos identificar os componentes de um exemplo. Essa tabela de exemplos refere-se a um domínio de cardiologia no qual o conjunto de pares atributo/valor das 6 primeiras colunas levam a conclusão das classes representadas na última coluna.

Sexo	Glicose	Stress	Pressão	Colesterol	Fumante	CLASSE
Masculino	Normal	Baixo	Normal	Normal	Não	Sem-predisposição
Masculino	Alta	Normal	Alta	Alto	Sim	Média-predisposição
Masculino	Alta	Baixo	Normal	Normal	Não	Pouca-predisposição
Feminino	Alta	Normal	Normal	Normal	Sim	Pouca-predisposição

Figura 6.2 - Tabela de exemplos

Um dos problemas dos métodos indutivos de AC é que eles são exclusivamente sintáticos (estatísticos) e por este fato, via de regra, geram resultados ineficientes a luz de um especialista do domínio. Para atenuar esse problema, batizado de problema semântico [Cirne91], obtemos uma certa quantidade de informações semânticas do especialista, na forma de relevâncias semânticas, que serão utilizadas juntamente com o conjunto de exemplos.

A relevância semântica indica a importância que tem um determinado par atributo/valor para concluir um elemento classificador. Essa importância é medida por associarmos um valor de pertinência, que indica o grau da relevância de um valor de um certo atributo para uma determinada classe. Esse valor está entre 0 e 1, onde 0 significa a ausência de relevância e 1 significa total crença na existência dela. Uma forma de representar a relevância semântica é através de uma matriz de relevância [Mongiovi90]. Ver exemplo da figura 6.3.

Classe / Atributo	Sem Predisposição	Pouca Predisposição	Média Predisposição	Muita Predisposição
Colesterol	0.8 /Normal	1.0/Normal	0.7/Alto	1.0/Alto
Fumante	0.6 /Não	0.7/Não	0.8/ Sim	0.7/ Sim
Glicose	0.7 /Normal			
Pressão	0.6 /Normal		0.7/ Alto	0.8/Alto
Sexo	0.1 /Masc.			
Stress	0.7 /Baixo	0.9/Baixo	0.1/Normal	0.8/Alto

Figura 6.3 - Matriz de relevância

Para a obtenção de exemplos devidamente estruturados e da relevância semântica sobre o domínio, devemos levar em consideração o tratamento de alguns problemas como: controlar a quantidade de

atributos, evitar atributos com valores contínuos, não permitir atributos sem valor e eliminar a ambiguidade (casos de exemplos duplicados, ou contra-exemplos).

6.2. Aplicação da metodologia FADO

Com base na descrição do domínio que fizemos na seção anterior, acompanharemos cada passo da metodologia FADO no desenvolvimento de uma solução para o problema de aquisição automática de conhecimento.

6.2.1. Análise do domínio da aplicação

Inicialmente, identificamos classes e objetos, atualizamos a biblioteca de classes, e depois, identificamos os estados e eventos.

6.2.1.1. Identificar classes e objetos

Este é o passo inicial de todo o processo de desenvolvimento iterativo. Nele, todas as classes do domínio do problema são identificadas. A partir das regras de identificação de classes e objetos, definidos no capítulo 5, seção 5.3.1, encontramos os seguintes objetos:

Por função:

Um domínio modelado : modelar o domínio da aplicação
Um tratador frequência : controlar quantidade de atributos
Um discretizador : evitar valores contínuo
Um tratador "buraco" : não permitir atributos sem valor
Um tratador ambiguidade: eliminar ambiguidade

Coisastangíveis:

Um exemplo
Um par atributo/valor
Um elemento classificador
Um algoritmo
Uma relevância

Locais:

Uma tabela de exemplos : conjunto de exemplos
Uma matriz de relevância: conjunto de relevâncias

Propriedade mensurável:

Uma pertinência : avalia o grau de incerteza da relevância

Procedimentos operacionais:

Uma ajuda : como já sabemos a interação homem-máquina é, hoje em dia, de grande valia para o sucesso de uma ferramenta. Por isto podemos logo optar pela utilização de uma classe com este propósito, em que seja especificado o nome de procedimento, o nível de autorização e a descrição dos passos a serem seguidos.

A partir dos objetos já conhecidos, partimos para identificar as classes. Existe uma diferença sutil entre classes e objetos. Enquanto um objeto representa uma entidade que existe no tempo e espaço, uma classe representa uma abstração [Booch 91]. As classes provém da análise de semelhanças das propriedades dos objetos. Por exemplo, um objeto *UmExemplo* faz parte da classe de objetos *Exemplo*, que fatora características inerentes a todos os exemplos.

Como resultado deste passo, pudemos traçar um primeiro esboço do diagrama estático e preencher os formulários de classes. A figura 6.4 mostra o esboço desse diagrama num nível mais alto de abstração.

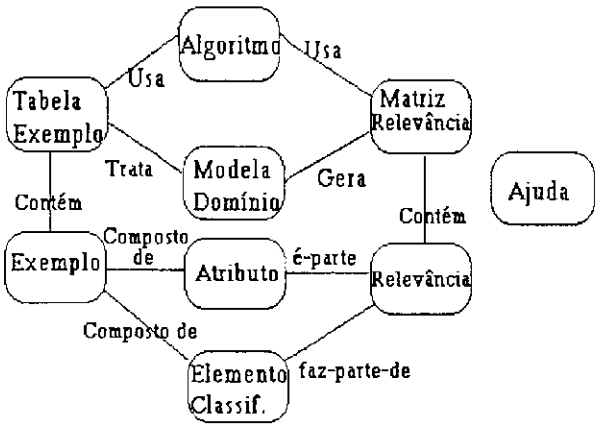

Figura 6.4 - Diagrama estático de alto nível

O passo de identificação de classes e objetos é finalizado com a colocação das classes identificadas na biblioteca de classes. As classes armazenadas foram: *Algoritmo, Relevância, Exemplo, Atributo, ElementoClassificador* e *Ajuda.* A classe *DomínioModelado* não deve ser colocada na biblioteca, por ser uma classe abstrata. Geralmente, as classes por função também não são boas candidatas, pois elas são específicas de uma aplicação.

As informações das classes a serem colocadas na biblioteca seguem o mesmo padrão do formulário de classe. O formulário de classe abaixo da classe *Exemplo* deverá ser refinado a medida que surgirem novos métodos, novas características e forem determinados os relacionamentos e abstrações entre as classes.

Classe	: Exemplo
Documentação	: É uma classe composta de pares atributo/valor para concluir um elemento classificador.
Métodos	
Instância	: IncluiExemplo,ExcluiExemplo,ConsultaExemplo

6.2.1.2. Identificar os estados dos objetos

Uma forma de auxiliar este passo é tentarmos responder a seguinte pergunta: quais os possíveis estados de um objeto que precisam ser guardados.

Para um objeto *Atributo* identificamos que seu estado é necessário ser lembrado no momento da estruturação e alimentação da tabela de exemplo. Isto acontece porque algumas críticas são feitas sobre um exemplo em função do estado dos atributos. Por exemplo, um exemplo será discretizado, se o estado do atributo for *ComValorContínuo* Os possíveis estados desse objeto são: *ComValorNominal,ComValor Contínuo, ComValorPoucoFrequente, ComValorRelevante e SemValor.*

6.2.1.3. Identificar os eventos

Para identificar os eventos partimos dos objetos já identificados e verificamos quais os eventos que causam alguma mudança no seu comportamento. As informações obtidas foram organizadas na tabela de evento da tabela 6.1.

Evento(s)	Tipo	Objetos Atingidos
InclusãoEx,ExclusãoEx	E	Exemplo
InclusãoRelev,ExclusãoRelev	E	Relevância,Atributo
InclusãoEICI,ExclusãoEICI	E	ElemClassif, Exemplo
InclusãoAtr,ExclusãoAtr,AlteraçãoAtr	E	Atributo, Exemplo
InclusãoAlg,ExclusãoAlg,ExecuçãoAlg	E	Algoritmo
Discretização	I	Discretizador, Atr, Ex
TratmBuraco	I	TratrBuraco, Atr,Ex
TratmFrequência	I	TratrFrequência,Atr,Ex
TratmAmbiguidade	I	TratrAmbiguidade,Atr,Ex

Tabela 6.1 - Tabela de eventos do problema

De posse dos estados dos objetos e dos eventos que atuam sobre eles, podemos construir os diagramas de transição de estados. A figura 6.5 é um exemplo do diagrama de transição de estado para o objeto *Atributo*. O formulário de cada evento é preenchido no momento em que o representamos neste diagrama.

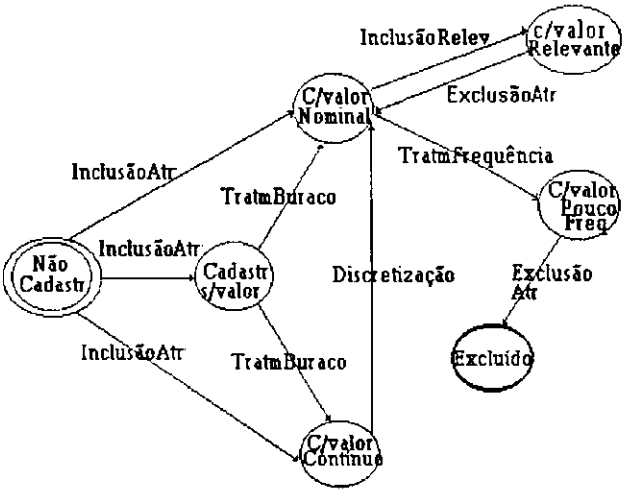

Figura 6.5 - Diagrama de transição de estado

6.2.2. Identificação da semântica das classes

Esta fase possui dois objetivos básicos. O primeiro objetivo se refere à tomada de decisões de projeto, considerando a semântica e relacionamento de cada classe. Vale salientar que as decisões de projeto

mais importantes (distribuição de funções) devem ser consideradas logo, adiando as menores (controle de restrições de integridade, de concorrência), que têm menos impacto no sistema. O segundo objetivo se refere à simplificação do projeto ao distribuir as funções entre as classes, considerando a comunalidade entre as abstrações. Esses objetivos devem ser alcançados ao passarmos pelos seguintes passos: projetar a solução do problema, identificar a semântica das classes e organizar suas estruturas. Esta fase é finalizada com a apresentação de um protótipo do projeto da solução.

6.2.2.1. Projetar a solução do problema

Neste passo propusemos um funcionamento que desse um suporte eficiente à modelagem do domínio. Em consequência, novas classes e eventos foram encontrados.

Funcionamento proposto

A nossa proposta foi fazer a modelagem do mundo real fundamentada em interações do especialista e/ou engenheiro de conhecimento com o sistema a ser desenvolvido, através de um entrevistador. O entrevistador ajuda a criar a estrutura de exemplos fornecendo sugestões e críticas. O usuário pode optar ou não pela presença do entrevistador. A opção por trabalhar sem o auxílio do entrevistador, só deve ser feita quando temos domínios, onde os exemplos já se apresentam com uma certa organização. O processo de entrevista, feito pelo ambiente ao usuário, é realizado através dos seguintes passos:

a. Perguntar sobre os elementos de classificação

O entrevistador pergunta quais os elementos de classificação que compõem os exemplos a cada nova tabela de exemplo. Os elementos de classificação podem ser aproveitados ao gerarmos uma tabela de exemplos a partir de uma já existente.

b. Estruturar a tabela de exemplo e criar a matriz de relevância

A aquisição de relevância semântica e a identificação dos atributos e valores que compõem um exemplo são realizadas nesta fase, com ajuda do entrevistador.

O entrevistador critica a existência de exemplos ambíguos e com atributos pouco frequentes. O usuário pode definir um número mínimo de exemplos incluídos, a ser tomado como medida para determinar quando serão feitas essas críticas, por exemplo, somente a partir de 10 exemplos incluídos e de um intervalo de 10 em 10 é que as críticas são feitas. O entrevistador também sugere a inclusão de elementos de classificação sem relevância. Essa sugestão é feita a cada intervalo de tempo de digitação definido pelo usuário.

c. Alimentar a tabela de exemplos

Neste momento são informados todos os casos que devem compor a tabela de exemplos, isto é, quais os valores dos atributos cadastrados no passo anterior, que determinam um elemento classificador. Em termos dos elementos de uma tabela, a estruturação corresponde à definição de colunas, e a alimentação, à inclusão das linhas.

d. Analisar a versão final da tabela de exemplos

Este passo consiste de uma análise final sendo feita a discretização e o tratamento de buracos. Estes últimos tratamentos são realizados antes da execução de um algoritmo. Eles são realizados por último porque qualquer mudança resulta em alterar toda a tabela de exemplos e, também, porque dependem das características do algoritmo executado.

Identificação de novas classes da solução

Uma vez descrito o processo de entrevista, que auxiliará na obtenção de entradas tratadas e na execução dos algoritmos, podemos identificar novas classes que colaboram para a realização desse processo. A definição das funções dessas novas classes deve ser feita considerando como as classes do problema e as da solução podem trabalhar juntas, para fornecer o comportamento esperado.

As classes da solução encontradas foram:

Entrevistador: responsável por entrevistar o especialista para identificar quais são os atributos, valores e elementos de classificação, que compõem a estrutura de um exemplo. Nesta entrevista são feitas sugestões e críticas para tratar as entradas.

Operador Interface: esta classe corresponde a classe *Usuário*, mencionada, geralmente, nas necessidades do sistema. O usuário é considerado fora do sistema, porque ele é simplesmente um agente que dispara eventos externos.

Gerente: esta classe é quase sempre responsável pela integração das classes, sendo sempre bem-vinda na solução de problemas. O gerente é responsável por verificar quais as características do algoritmo instanciado para ativar os objetos responsáveis para um tratamento adequado, bem como gerenciar a alimentação dos exemplos. Ele contém informações do ambiente, como: associação de matrizes à tabelas de exemplos, opção por um entrevistador e por versão automática, o número mínimo de exemplos incluídos para serem ativadas as críticas e o tempo de digitação para serem ativadas as sugestões.

Identificação dos eventos da solução

Como não poderia deixar de ser, novos objetos e novos procedimentos, causam a descoberta de novos eventos. Assim, fizemos o refinamento da tabela de eventos da tabela 6.1 e a atualização do diagrama de transição de estado da figura 6.5, incluindo os eventos da solução mostrados na tabela 6.2. Há casos em que já temos os eventos e descobrimos objetos. A descoberta do objeto *Tratador Composição*, a partir do evento *Composição*, exemplifica como pudemos descobrir objetos, até então despercebidos, a partir de eventos que atuaram sobre ele.

Evento(s)	Tipo	Objetos Atingidos
PedidoNovo, Seleção, SalvarComo	E	TabelaExemplo
TempoDigitação	T	Entrevistador
Composição	E	TratrComposição,Atr,Ex

Tabela 6.2 - Tabela de eventos da solução

6.2.2.2. Identificar a semântica dinâmica das classes

Neste passo colocamos os métodos em suas classes apropriadas, construindo a tabela de ativação entre objetos da tabela 6.3.

Ob.Pass	Ob.Ativo	Método	Evento(s)
O.I.	Exemplo	IncluiEx	InclusãoEx
O.I.	Exemplo	ExcluiEx	ExclusãoEx, ExclusãoEICI,TratmAmbiguidade
O.I.	TabelaExemplo	IniciaTE	PedidoNovo
S.I.S.	TabelaExemplo	ReorganizaExs	Discretização, TratmBuraco, ExclusãoAtr, AlteraçãoAtr
Exemplo	TabelaExemplo	AssDessTE	InclusãoEx, ExclusãoEx
S.I.S.	TabelaExemplo	SalvaVersão	SalvarComo, Composição, Discretização, TratmBuraco, XExDig
S.I.S.	TabelaExemplo	ConsultaExs	TratmAmbiguidade,TratmFrequência,ExecuçãoAlg
O.I.	Relevância	IncluiRelev	InclusãoRelev
O.I.	Relevância	ExcluiRelev	ExclusãoRelev
Algoritmo	MatrizRelevânc	ConsultaRels	ExecuçãoAlg, TempoDigitação
Relevância	MatrizRelevânc	AssDessMR	InclusãoRelev, ExclusãoRelev
TabEx	Entrevistador	MostraCrítica	PedidoNovo
TrtrFreq	Entrevistador	MostraCrítica	TratmFrequência
TrtrAmb	Entrevistador	MostraCrítica	TratmAmbiguidade
C.A.	Entrevistador	SugereRelev	TempoDigitação
O.I.	Algoritmo	IncluiAlg	InclusãoAlg
Algoritmo	Gerente	AssAlgTeMr	InclusãoAlg
O.I.	Algoritmo	ExcluiAlg	ExclusãoAlg
Algoritmo	Gerente	GerenciaExAlg	ExecuçãoAlg
Exemplo	Gerente	GerenciaAlim	InclusãoEx, AlteraçãoAtr, ExclusãoEICI, ExclusãoAtr
Gerente	TratadorBuraco	TrataBuraco	TratmBuraco
Gerente	Discretizador	TrataDiscr	Discretização
Gerente	TratadorFrequência	TrataFreq	TratmFrequência
Gerente	TratadorAmbiguidade	TrataAmb	TratmAmbiguidade
O.I.	TratadorComposição	CompõeAtr	Composição

Tabela 6.3 - Tabela de ativação entre objetos

Onde:

O.I. – OperadorInterface

S.I.S - sem informações suficientes para definir

C.A. - controle automático feito pela classe *Regra* pré-definida no TOM.

O leitor pode questionar porque a classe *TabelaExemplo* foi utilizada neste nível de abstração, para os métodos *ReorganizaExs* e *ConsultaExs* e não diretamente a classe *Exemplo*? A resposta é que o uso do objeto *TabelaExemplo* dá mais significância lógica e física a um agrupamento de exemplos. E também, porque há a necessidade de reorganizar todos os exemplos ou consultá-los, em reação a algum dos eventos citados. Estas operações são realizadas de forma iterativa sobre a tabela, significando que pode ser feito um acesso a cada exemplo e em qualquer ordem.

6.2.2.3. Organizar a estrutura das classes

Em algumas situações, a aplicação de generalização muda a semântica das classes ao criar novas e/ou eliminar algumas já existentes. No diagrama estático da classe *DomínioModelado* da figura 6.6, encontramos casos de generalização. Observe que as classes *Discretizador, TratadorFrequência , Tratador Composição , TratadorBuraco* e *TratadorAmbiguidade,* foram generalizadas na classe *Otimizador.*

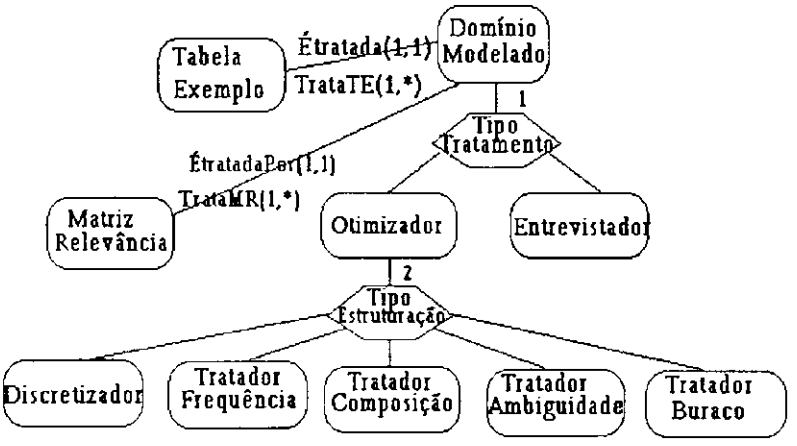

Figura 6.6 - Diagrama estático de DomínoModelado

A partir das heurísticas definidas em [Yourdon92] pudemos eliminar as classes *Discretizador, TratadorBuraco, TratadorFrequência, TratadorAmbiquidade* e *Tratador Composição*, transformando-as em métodos de sua superclasse. Essas classes deixaram de ser classes pelos seguintes motivos: é inconcebível a existência de uma classe que só tenha uma única instância, só possua uma única função e não possua atributos.

Após esta modificação tivemos que alterar a tabela de ativação entre objetos da tabela 6.3. Além disso, a interface de cada classe é atualizada pela identificação de novos métodos. Na verdade essa identificação acontece durante todo o restante do processo de desenvolvimento refletindo o seu caráter incremental.

6.2.2.4. Apresentar um protótipo

O domínio que estudamos tem um caráter original, pois não há nenhum sistema já definido de AC automática, não sendo possível aproveitar nenhuma classe. Mas nós pudemos traçar um pequeno

protótipo (figura 6.7) da solução encontrada para estruturar a tabela de exemplos. Os pares atributos/valor e os elementos de classificação são entrados nas suas respectivas janelas. Quando os atributos são incluídos eles vão automaticamente para a janela de tabela de exemplos. Depois, os valores de atributos, que farão parte da tabela de exemplos, são selecionados da lista de pares atributos/valores.

Para a construção deste sistema, contamos com a classe *Interface*, que nos possibilitou abstrair do controle dos menus, ícones, janelas, mouse e listas.

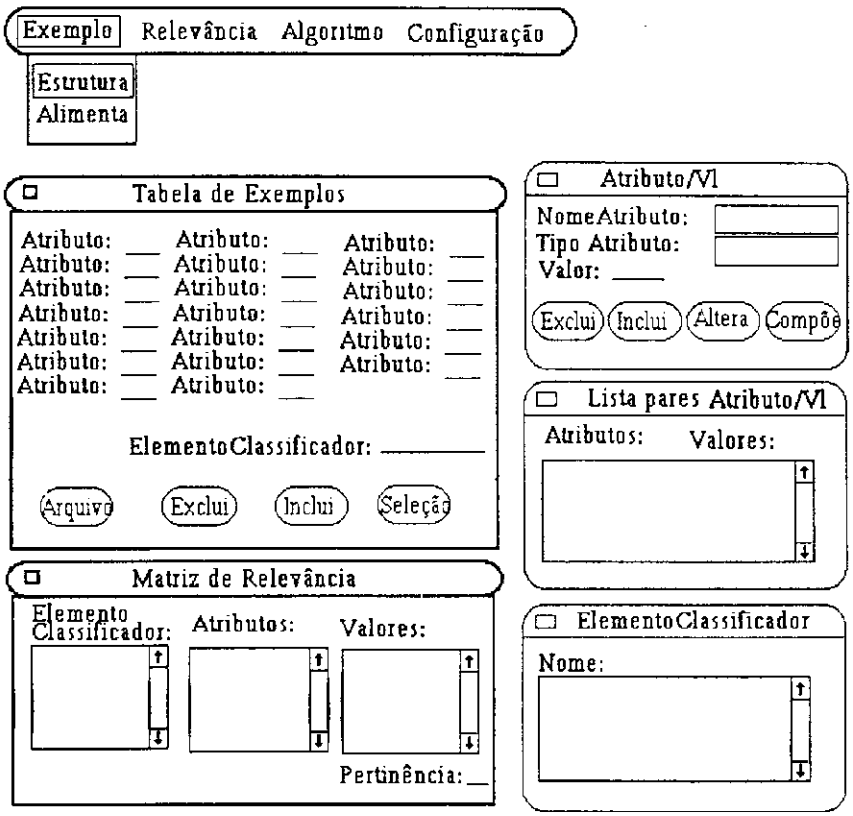

Figura 6.7 - Protótipo

6.2.3. Identificar os relacionamentos entre classes

Nesta fase representamos os relacionamentos, as abstrações entre classes e as características temporais e de versionamento existentes no modelo TOM.

6.2.3.1. Identificar os relacionamentos e abstrações

Neste passo devemos refinar os diagramas estáticos com informações que dizem respeito a interface da classe. Essas informações se referem às seguintes características:

Formas de abstrações entre classes

As formas da abstração aplicadas nos diagramas estáticos são: generalização, agregação e agrupamento. Cada uma dessas abstrações possuem formulários a serem preenchidos no momento em que relacionamos duas classes de um diagrama estático. A ligação varia conforme a abstração:

[A] Generalização

A ligação entre a superclasse e a subclasse é representada pelo desenho de um *losango*. Neste momento o formulário de generalização deve ser preenchido. Vejamos o formulário abaixo preenchido para a classe *DomínioModelado* da figura 6.6:

NumAbstração	: 1
Superclasse	: DomínioModelado
Subclasses	: Otimizador, Entrevistador
Papel	: TipoTratamento
Parâmetros	: Disjuntivo,Completo
TipoGeneralização	: Explícito

[B] Agregação

A ligação entre a classe agregada e seus componentes é representada pelo desenho de um *quadrado*. As características da agregação da classe *Relevância*, representada no diagrama estático de *Relevância* da figura 6.8, são mostradas no formulário de agregação a seguir.

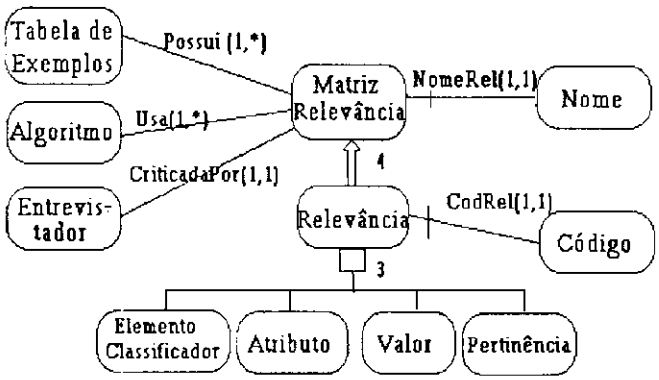

Figura 6.8 - Diagrama estático de Relevância

NumAbstração	: 3
Agregada	: Relevância
Componentes	: Atributo, ElementoClassificador, Valor e Pertinência
TipoAgregação	: Explícito
Herança de Método	: (IncluiRelev, Computada (IncluiElCl, IncluiAtr, IncluiPertinência))

[C] Agrupamento

A representação de um agrupamento é feita por uma ligação direta entre a classes grupo e a classe elemento. Para o agrupamento entre as classes *MatrizRelevância* e *Relevância* da figura 6.8, mostramos o preenchimento do formulário de agrupamento.

NumAbstração	: 4
Grupo	: MatrizRelevância
Elemento	: Relevância
Parâmetros	: Completo, OrdenadoPor (:Atributo)
TipoAgrupamento	: Explícito

Relacionamentos de classe e instância

No diagrama estático da classe *Algoritmo* da figura 6.9, encontramos somente relacionamentos de instâncias. O relacionamento de instância *NomeAlg* relaciona a instância da classe *Algoritmo* com a instância da classe *Nome*.

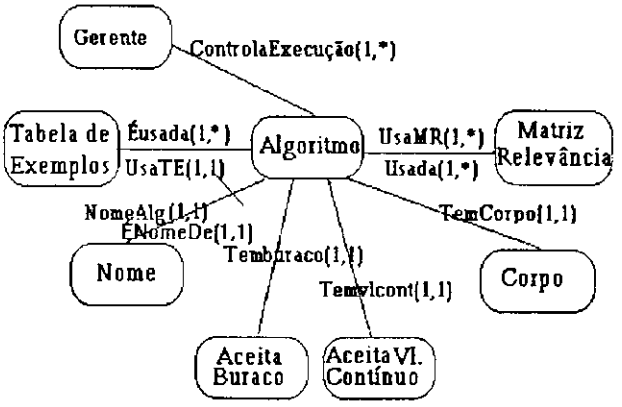

Figura 6.9 - Diagrama estático de Algoritmo

TipoRelacionamento	: Instância
ClasseOrigem	: Algoritmo
Relacionamento	: NomeAlg
Cardinalidade	: 1:1
ClasseDestino	: Nome
RelacionamentoInv	: É-nome-de
Cardinalidade	: 1:1
ChaveParaClasse	: Sim

No diagrama estático da classe *Exemplo* da figura 6.10 são apresentadas características temporais e de versionamento que descreveremos a seguir. Ainda nesta figura encontramos duas metaclasses (G_CLASS e V_CLASS), vários relacionamentos de instâncias, abstrações e um relacionamento de classe entre as classes *Exemplo e Quantidade,* denominado *MaxAtrEx.O* formulário de relacionamento (RC/RI) para esse relacionamento de classe, deve ser preenchido no momento em que há a ligação entre as classes *Exemplo* e *Quantidade.*

TipoRelacionamento	: Classe
ClasseOrigem	: Exemplo
Relacionamento	: MaxAtrEx
Cardinalidade	: 1:1
ClasseDestino	: Quantidade
ChaveParaClasse	: Não

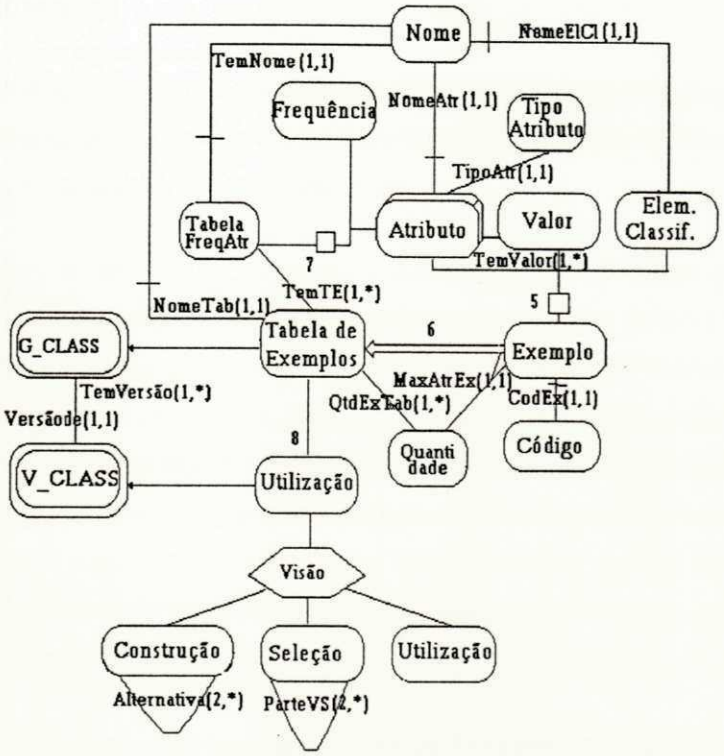

Figura 6.10 - Diagrama estático de Exemplo

Cardinalidade e limitações de valores

Tivemos que tomar decisões de projeto com relação a cardinalidade dos relacionamentos e limites de valores das classes. Por exemplo, especificamos que uma tabela de exemplos pode ter mais de uma matriz de relevância e cada exemplo contém até 20 pares atributo/valor.

Classes temporais

Pela figura 6.10 percebemos que a classe *Atributo* foi definida como temporal, para nos permitir guardar todas suas instâncias excluídas. Classes temporais foram definidas no projeto quando buscamos representar a relevância semântica, e se tornou necessário manter as informações históricas de atributos relevantes. No formulário da classe *Atributo* preenchido abaixo, definimos que guardamos as informações durante 6 meses. Após este período, elas são removidas, definitivamente, das respectivas classes. Antes desse período, o usuário pode recuperar o atributo.

Classe	: Atributo
Documentação	: Esta classe faz parte de um exemplo e de uma relevância
Tipo da classe	
Temporal	: Sim
Tempo	: 00:06:00:00:00:00
Relacionamentos Instância	: NomeAtr:Nome, TipoAtr:TipoAtributo,TemValor:Valor
Métodos Instância	: IncluiAtr,ExcluiAtr,AlteraAtr,ConsultaAtr

Classes versionadas

Em aplicações de ambientes de desenvolvimento de software, guardar a evolução da construção dos objetos é de grande valia para o projetista. Para atender a esta necessidade o TOM fornece um mecanismo de controle de versões de objetos no banco de dados.

Como já foi dito no início deste capítulo, nosso estudo de caso é um CASE de sistemas baseados em conhecimentos. Uma das principais tarefas do ambiente é modelar o domínio em exemplos. Esses exemplos passam por várias transformações até obtermos uma tabela consistente. Portanto, definimos a classe *TabelaExemplo* como versionada (figura 6.10). Manter as diferentes versões de uma tabela de exemplo é importante, porque os algoritmos estatísticos podem ser aplicados a algumas dessas versões. Com isto, podemos fazer análises comparativas entre os resultados obtidos para descobrir qual a entrada mais eficiente. Como já dissemos ao especificar o TOM no capítulo 3, para representar uma classe versionada devemos defini-la como subclasse da metaclasse G_CLASS, e suas versões individuais como subclasses da metaclasse V_CLASS.

Para verificarmos a evolução das diversas versões da *TabelaExemplo*, devemos seguir todo o seu o ciclo de vida e determinarmos os eventos que requerem a geração de versão. Elas passam por 3 estados ou visões distintas:

a. Construção - esta fase corresponde aos diversos momentos em que a tabela é estruturada e alimentada. As versões automáticas são feitas a cada "X" exemplos incluídos na tabela (esta quantidade é definida pelo usuário) ou quando ocorre o evento externo *composição* . Esse evento se refere a junção de dois atributos num único atributo. Os valores do atributo fundido são determinados de duas maneiras diferentes: eles podem ser resultantes de uma análise combinatória entre os valores dos atributos ou somente a soma dos valores dos dois atributos. O especialista pode escolher qualquer uma dessas alternativas, No gráfico de versão da figura 6.11, a versão 3.1 não foi bem sucedida, voltando à versão 3.2 para iniciar a seleção dos exemplos.

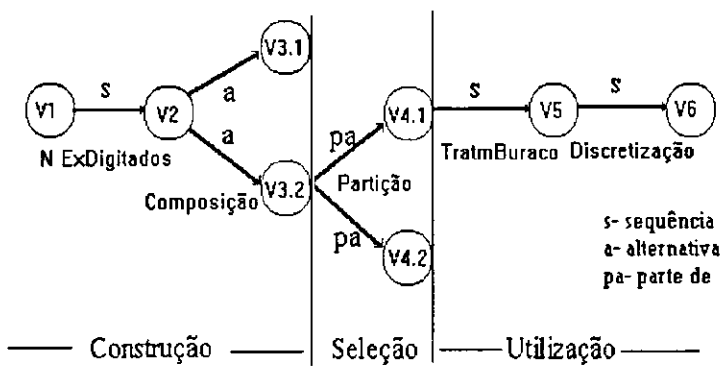

Figura 6.11 - Gráfico de versão

b. Seleção – este é a fase em que é escolhido uma tabela de exemplos ou parte dela, para execução do algoritmo. Podem ocorrer partições da tabela de exemplos original e por isto diferentes versões podem ser geradas para cada parte.

c. Utilização – nesta fase a tabela de exemplos é utilizada pelo algoritmo. Mas antes da execução do algoritmo, a tabela de exemplos selecionada pode ser alterada, diante dos eventos de tratamento de buracos e discretização, por isto uma nova versão a cada um desses eventos é guardada.

Relacionamentos temporais

Esta situação ocorre quando há a necessidade de manter os valores, as datas de início e fim, dos relacionamentos anteriores do relacionamento presente.

Na figura 6.12 mostramos um relacionamento temporal entre as classes *Gerente* e *VersãoAutomática*, para controlar se havia opção automática para gerar versão da tabela de exemplo numa determinada época. Esta informação é útil para efeito de histórico, segurança e consistência.

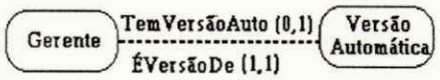

Figura 6.12 - Relacionamento temporal

Consideremos o formulário de relacionamento (RC/RI) abaixo, preenchido com as informações desse relacionamento temporal. No caso serão guardadas até 20 mudanças de opção.

TipoRelacionamento	: Instância
ClasseOrigem	: Gerente
Relacionamento	: TemVersãoAuto
Cardinalidade	: 0:1
ClasseDestino	: VersãoAutomática
RelacionamentoInv	: ÉVersãoDe
Cardinalidade Inv	: 1:1
ChaveParaClasse	: Não
QtdValorAnterior	: 20

6.2.3.2. Identificar os relacionamentos dinâmicos

Neste passo pretendemos descobrir *quem pede o que* e *a quem*. Como ferramenta de apoio neste passo, utilizamos a tabela de ativação entre objetos (tabela 6.3), preenchendo os objetos ativos que ainda não tinham sido preenchidos. Essa tabela dá suporte à construção do diagrama contextual da figura 6.13. Vale salientar, que as classes desse diagrama são aquelas do nível mais alto de abstração dos respectivos diagramas estáticos. Os métodos disparados pelos eventos externos com relação as classes *Atributo e Elemento Classificador* não foram representados, para efeito de clareza desse diagrama. Esses métodos foram descritos num nível de abstração mais baixo, quando construímos o diagrama dinâmico de *Exemplo*.

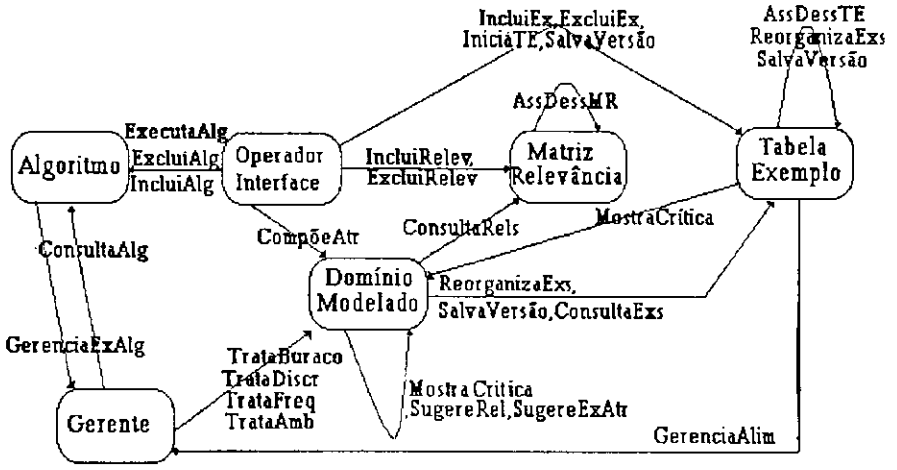

Figura 6.13 - Diagrama contextual

O formulário de métodos é preenchido ao fazermos as ligações entre as classes. Vejamos o modelo descrito a seguir do método *ReorganizaExs* da classe *TabelaExemplo*. Os parâmetros utilizados são: *Op* que pode ser "A" para alteração ou "E" para exclusão, *Tp* que pode ser "At" para Atributo ou "ElCl" para Elemento Classificador, *Nom* que é o nome do atributo ou elemento classificador utilizado, *Nov* onde é colocado um novo nome no campo quando a opção for de alteração. No corpo da ação sabemos que a *TabelaExemplo* deve ser toda percorrida para excluir exemplos ou reorganizá-los, mas não definimos ainda se será usado *trigger* para ativar estas ações.

Método	: ReorganizaExs
Documentação	: Reorganiza todos os exemplos da classe Exemplo, que estão na tabelaExemplo
Classe do método	: TabelaExemplo
Tipo do método	: Instância
ParâmetrosEntrada:	(Op:Caracter, Tp:String, nom:String, nov:String)

6.2.3.3. Identificar os relacionamentos pré-pós

Determinamos relacionamentos pré-pós para uma classe, observando os estados contidos no seu diagrama de transição de estado. Ao preenchermos o formulário de classe devemos optar ou não por gerar esses relacionamentos.

6.2.4. Análise comportamental das classes

Neste ponto, abordamos as características de projeto que foram adiadas na fase 6.3, como: restrições de integridade, concorrência e controle de eventos.

6.2.4.1. Identificar restrições

No TOM, para que um método seja executado, é necessário que as condições de *trigger* e/ou as

pré-condições de método sejam aceitas. Elas representam o controle de integridade sobre o banco de dados. Para descobri-las aconselhamos analisar as condições de ocorrência dos eventos internos e temporais. Na tabela 6.4, vemos as condições dos eventos das tabelas 6.1 e 6.2 e mais dos eventos *ExemploIncluído* e *Reorganização*. Esses eventos foram identificados pela necessidade de associarmos condições às ações ao serem chamadas pelos métodos que gerou o evento.

Evento	Tipo	Condição de ocorrência do evento
Discretização	I	Ocorre antes da execução do algoritmo e se tipo atributo é contínuo
TratmBuraco	I	Ocorre antes da execução do algoritmo e se não há valor no atributo
TratmAmbiguidade	I	A cada X exemplos digitados e entrevistador ativo
TratmFrequência	I	A cada X exemplos digitados e entrevistador ativo
ExemploIncluído	I	A cada atributo e EICI cadastrado, seleciona-os para formar exemplo
Reorganização	I	A cada atributo e EICI excluído, atributo alterado, discretização
TempoDigitação	T	A cada 30 min de digitação e entrevistador ativo sugere relevância

Tabela 6.4 - Tabela de eventos c/ condições

De acordo com as informações acima, vemos que antes da execução de um algoritmo, acontecem os eventos *discretização e tratmBuraco*. Como eles ocorrem no mesmo instante, passamos a representar um único evento, denominado *DiscrBuraco*. As condições que os diferem são colocadas no *trigger*. O mesmo ocorre para os eventos *tratmFrequência e TratmAmbiguidade*, nos quais chamamos agora de *TratmFreqAmb*

Como já dissemos, as condições do *trigger* podem ser retiradas da condição de ocorrência dos eventos. Por exemplo, para que o evento interno *ExemploIncluído* aconteça, os *Atributos e EICI* já devem estar cadastrados. Esta condição será do *trigger*, que dispara as ações que seleciona vários pares atributo/valor (ConsultaAtr) e um elemento de classificação para fazer parte de um exemplo (ConsultaEICI) e depois associa esse exemplo incluído à tabela de exemplos (AssDessTE).

Voltemos agora ao formulário do método *ReorganizaExs* (abaixo da figura 6.13) que deixamos pendente a definição do corpo da ação. Podemos dizer que a condição que define que método deve ser ativado *ExcluiEx* ou *ReorganizaEx* , será a condição de um *trigger* e esses métodos, como são mutuamente exclusivos, serão considerados a ação e a falha-ação, respectivamente. No corpo da ação do método *ReorganizaExs* deverá ficar um comando de ativação desse *trigger* (no caso,T3). Vejamos a seguir os formulários do *trigger* T3 e o da sua condição. (As definições de todas as informações dos *trigger* s utilizados neste sistema são encontradas no apêndice A.).

Trigger	: T3
Habilitado	: Sim
Condição	: C3
Ação	: ExcluiEx
F_Ação	: ReorganizaEx

NoCondição	: C3
Documentação	: testa se parâmetros são exclusão de EICI
Estrutura Sentença	: tp = "EICI" and op = "E"
Falha	: Avise
Mensagem	: "A tabela de exemplos foi reorganizada"

A ativação do método por um evento externo pode gerar um evento interno, que dispara triggers ou simplesmente uma chamada a uma sub-rotina. Existem dois motivos pelos quais uma ação é chamada via *trigger* ou não. Primeiro, é a existência de condições para a ocorrência de uma ou mais ações. Segundo, é a existência de ações, denominadas *falha-ação*, que são executadas quando essas condições falham. Ainda neste passo, fazemos o refinamento dos formulários de métodos, por definirmos as condições associadas a eles.

6.2.4.2. Associar eventos a triggers

A partir da tabela de ativação entre objetos e das informações dos *triggers*, geramos os formulários de regras.

Regra:R1 Evento:ExemploIncluído Triggers:T1,T2	Regra:R2 Evento:Reorganização Triggers:T3	Regra:R3 Evento:TratmFreqAmb Triggers:T4,T5

Regra:R4 Evento:TempoDigitação Triggers:T6	Regra:R5 Evento:DiscrBuraco Triggers:T7,T8

Os formulários de eventos são refinados ao colocarmos as regras associadas a eles. Vejamos abaixo esse formulário para o evento temporal *TempoDigitação* que ocorre após 30 min de exemplos incluídos.

Evento	: TempoDigitação
Documentação	: critica a tabela a cada 30min
Ativo	: Sim
Regra	: R4
TipoEvento	: Temporal
Quando	: 00:00:00:00:30:00
Atraso	: 00:00:00:00:00:00
Tipo	: discreto
TipoEvTemporal	:Relativo
EventoRelativo	:ExemploIncluído

6.2.4.3. Representar o comportamento dinâmico entre as classes

Neste passo construímos os diagramas dinâmicos das classes existentes no diagrama contextual da figura 6.13 integrados aos diagramas estáticos das respectivas classes. As informações das regras ativas: *triggers* e eventos, são encontradas no apêndice A.

Na figura 6.14 mostramos o diagrama dinâmico da classe *Exemplo*, juntamente com suas classes componentes (*Atributo* e *ElementoClassificador)* e sua classe grupo (*TabelaExemplo*). As outras classes (*Gerente, Otimizador* e *Entrevistador*) foram representadas para mostrar como elas se relacionam com as classes que não são parte de sua estrutura. Neste diagrama existem três *triggers* T1, T2 e T3, e vários eventos externos e internos. Existe também, um grafo de dependência para o evento *Reorganização*, significando que para que ele aconteça é necessário que qualquer um dos eventos: *ExclusãoAtr*(EA),

ExclusãoElCl (EE), *AlteraçãoAtr* (AA) ou *TratmDiscrBuraco* (DB) ocorram. Estas informações foram retiradas da coluna *condição de ocorrência do evento* da tabela 6.4.

Na figura 6.15 mostramos o diagrama dinâmico de *Relevância* e de sua classe grupo *MatrizRelevância* representando a troca de mensagens entre elas e entre as classes *Atributo, ElementoClassificador, Algoritmo* e *Entrevistador.* Neste diagrama existem dois eventos externos que disparam os métodos IncluiRelev e ExcluiRelev. O método *ConsultaRels* de *MatrizRelevância* , que é invocado por qualquer uma das classes *Algoritmo* ou *Entrevistador,* consulta todas as instâncias de *Relevância.*

O diagrama dinâmico da figura 6.16 é da classe *DomínioModelado.* Nele aparece a especificação de suas subclasses: *Entrevistador* e *Otimizador* e as classes com que estas se relacionam. Os eventos que causam a ativação dos métodos são: *Composição, TempoDigitação, TratFreqAmb* e *TratDiscrBuraco.* Os outros eventos fazem parte somente do grafo de dependência entre eventos. Por exemplo, vemos que o evento *TempoDigitação* depende da ocorrência do evento *ExemploIncluído* para ocorrer.

O diagrama dinâmico da figura 6.17 são das classes *Algoritmo* e *Gerência.* Elas foram colocadas num mesmo diagrama por questões de simplicidade, já que elas não se relacionam com muitas classes, e, principalmente, por elas se relacionarem.

6.2.4.4. Identificar concorrência entre os métodos

Neste passo devemos refinar os formulários de triggers colocando a ordem de execução das ações e a prioridade. Por exemplo, associada à regra R1, estão os triggers T1 e T2. Como as ações de T1 devem ser executadas primeiro que as ações de T2, T1 deve ter prioridade maior que T2.

6.2.4.5. Apresentar um protótipo

O protótipo anterior da estrutura e alimentação da tabela de exemplos foi bem aceito, havendo somente pequenos ajustes na apresentação das informações e novas janelas para atualização da relevância, dos algoritmos e da configuração do ambiente em uso.

6.2.5. Implementação de classes

Este passo requer um nível de detalhamento maior ao especificar o corpo da ação de todos os métodos dos formulários de métodos. A definição do corpo da ação do método *ReorganizaExs* na *TabelaExemplo* está de acordo com a sintaxe do TOM, e significa a ativação do *trigger* T3, enquanto existirem exemplos na *TabelaExemplo.*

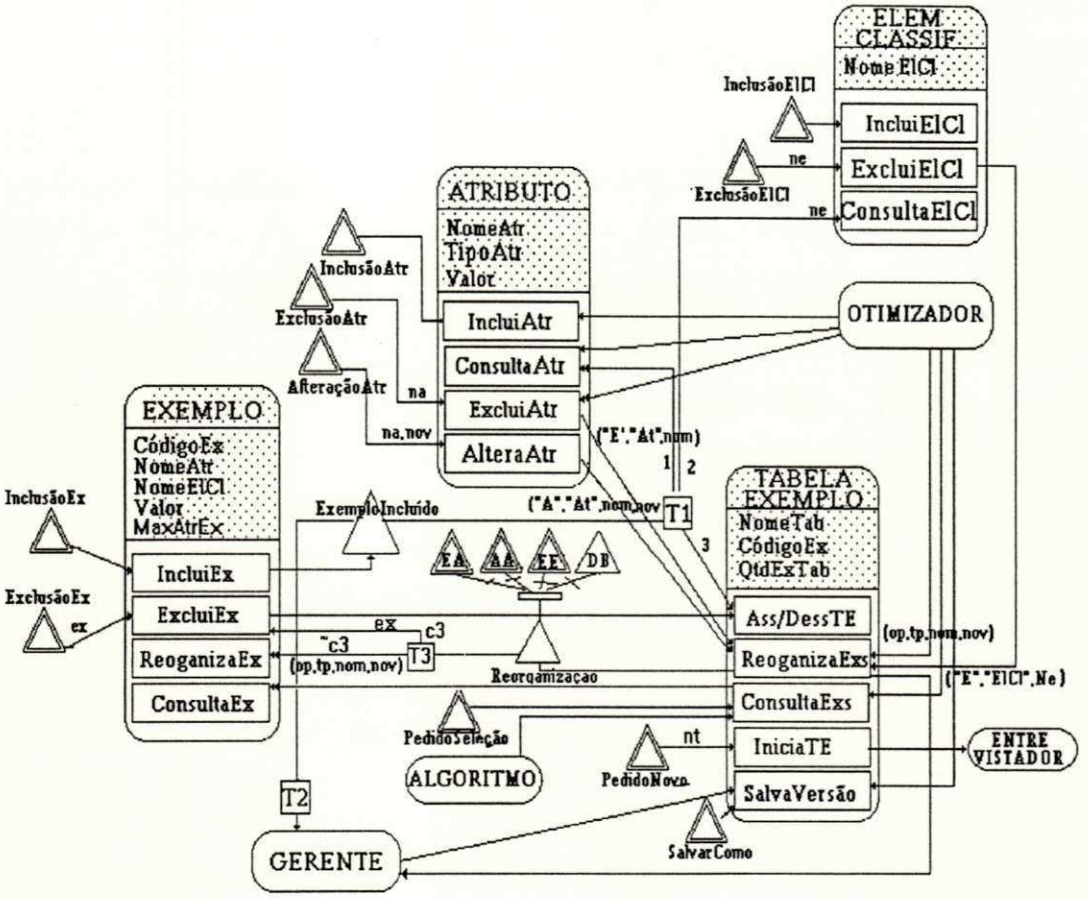

Figura 6.14 - Diagrama dinâmico de Exemplo

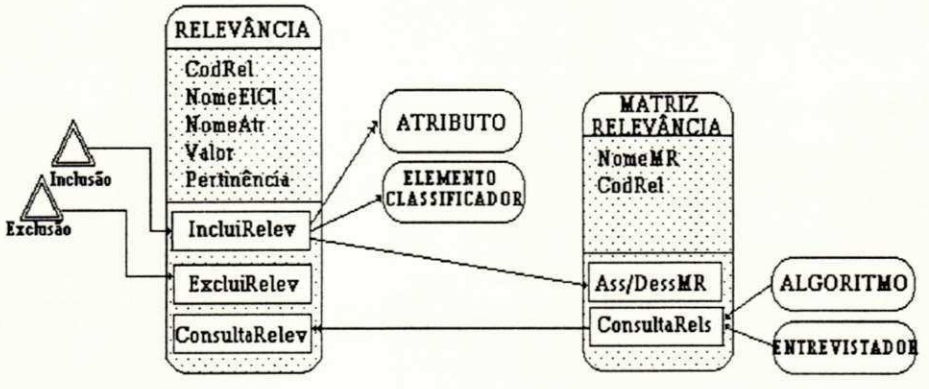

Figura 6.15 - Diagrama dinâmico de Relevância

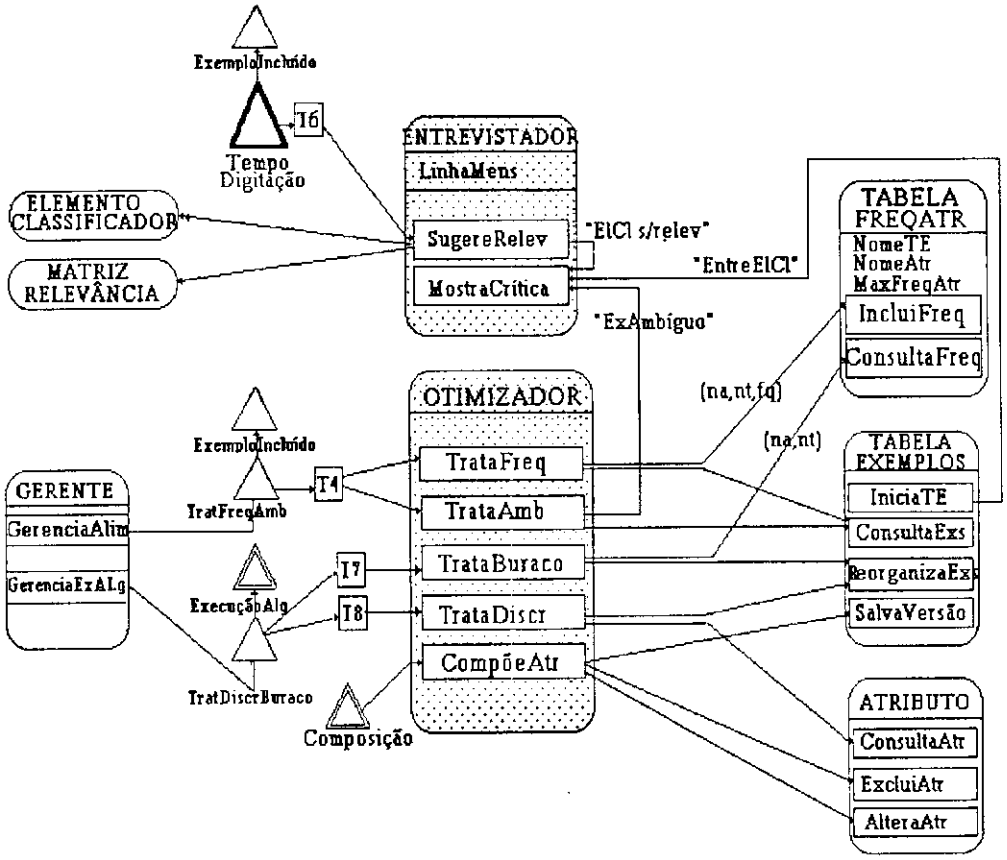

Figura 6.16 - Diagrama dinâmico de DomínioModelado

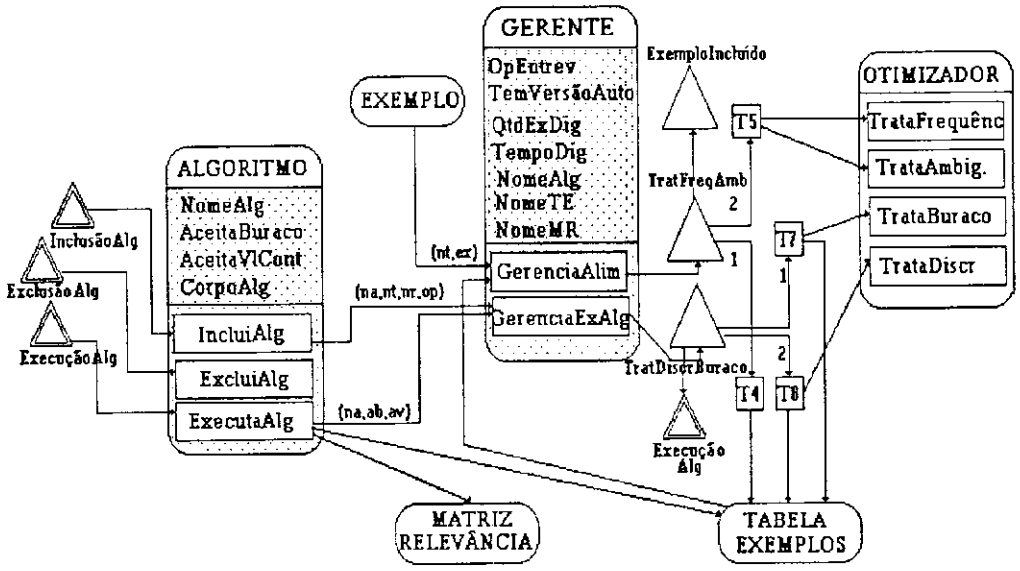

Figura 6.17 - Diagrama dinâmico de Gerente e Otimizador

Método	: ReorganizaExs
Documentação	: Reorganiza todos os exemplos da classe Exemplo, que estão na tabelaExemplo
Classe do método	: TabelaExemplo
Tipo do método	: Instância
ParâmetrosEntrada	: (Op: Caracter, Tp:String, nom:String, nov:String
CorpoDaAção	: For each X:Exemplo in TabelaExemplo DO T3

6.2.6. Transformação do esquema conceitual no esquema lógico

Neste passo, a partir das informações contidas nos formulários e nos diagramas geramos o esquema lógico na sintaxe TOM, o qual é encontrado no apêndice A. Um suporte automatizado à obtenção da definição de classes no modelo de dados TOM, a partir dos diagramas de estruturas e comportamentos, torna mais ágil e eficiente a tarefa de especificação do domínio da aplicação.

Capítulo 7

Ferramenta FADO

7.1. Introdução

As ferramentas CASE (Computer Aided Systems Engineering) surgiram para dar aos projetistas suporte às tarefas e técnicas complexas em quase todas as fases do ciclo de vida. Possibilitando obter no final do desenvolvimento do produto todas as informações coletadas devidamente especificadas e o sistema documentado.

Utilizar eficientemente uma ferramenta CASE, doutrina o projetista a seguir uma metodologia e ajuda-o a obter um produto de qualidade. Consequentemente, se terá uma redução de custos, por fazer uma melhor administração do tempo de produção. Devido a importância dessas ferramentas propomos um projeto de interface de uma ferramenta CASE para a metodologia FADO. Esse projeto ainda não está implementado, mas objetiva mostrar o projetista as facilidades que ele pode ter ao utilizar a metodologia com apoio automatizado, como: análise de inconsistências, integração dos modelos estáticos e dinâmicos, reutilização das classes e geração automática do esquema lógico TOM. Ao projetarmos essa interface consideramos o objetivo da ferrramenta, o tipo de usuário que a utilizará e os recursos disponíveis para confeccioná-la.

Neste capítulo, veremos a descrição funcional da ferramenta, o tipo de usuário destinado, os tipos de diálogos utilizados e o projeto das telas de menus.

7.2. Objetivos e funcionalidades da ferramenta FADO

A ferramenta FADO auxilia o projeto de um banco de dados na representação estrutural e comportamental dos objetos do domínio e no fornecimento da especificação das classes no modelo de dados TOM. A ferramenta FADO apóia a construção do esquema conceitual de aplicações orientadas a objetos.

Na figura 7.1 podemos ver que a ferramenta acompanha o processo FADO, na obtenção das informações do formulário, colocando-as num único dicionário, na confecção dos diagramas e na geração do esquema lógico do modelo de dados TOM.

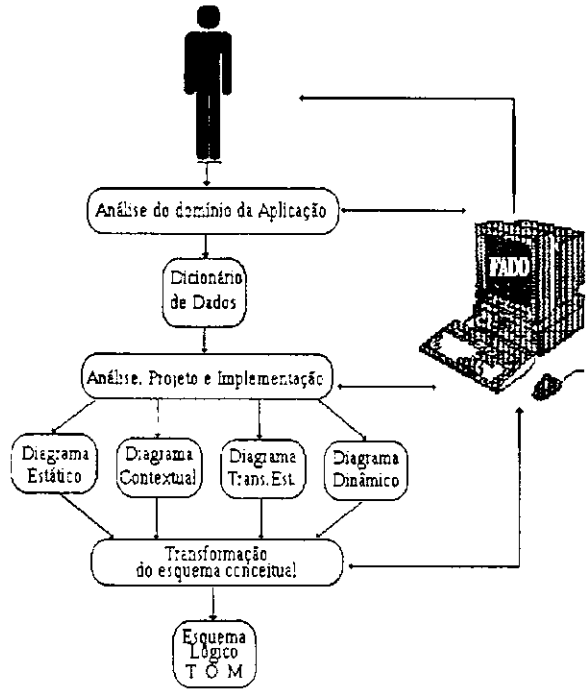

Figura 7.1 - Arquitetura da ferramenta FADO

As funções da ferramenta FADO são:

- Dar suporte à metodologia FADO, desde a especificação de requisitos até a implementação, realizando checagem de consistência de análise contextual e fornecendo controle das informações sobre um único dicionário de dados;
- Oferecer recursos para construção e otimização de diagramas compostos de figuras e ligações entre elas;
- Receber as informações dos elementos do dicionário de dados;
- Gerar o esquema lógico TOM a partir dos diagramas e informações do dicionário;
- Gerar o relatório de inconsistências a partir dos diagramas e do dicionário;
- 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 as fases da metodologia e como construir os diagramas.

7.3. Característica do Usuário

No modelo de dados TOM existem dois níveis de abstração: o metanível onde o administrador de modelos define a sintaxe do modelo e o nível de aplicação onde o projetista define a sintaxe de um problema específico. A ferramenta FADO é utilizada tanto para definição do esquema conceitual das metaclasses do metanível, como para definição das classes do nível de aplicação. As informações contidas no metanível, devem ser modificadas somente pelo administrador de modelos.

Uma vez que o administrador de modelos defina o esquema conceitual do metanível, ele só utilizará

de novo essa ferramenta diante de alguma modificação nas metaclasses. O mesmo acontece ao projetista com relação ao esquema conceitual da aplicação. Como esses dois usuários são ocasionais, a ferramenta deve ser simples, contendo estilos de diálogos apropriados, e métodos de ajuda *on-line*.

7.4. Descrição da interface da ferramenta FADO

7.4.1. *Software* proposto

O projeto dessa interface foi influenciado por características do aplicativo Design/OA[Metasoft89]. Nós escolhemos esse aplicativo porque ele fornece um ambiente aberto para a construção, manipulação e análise de diagramas, sem o programador se preocupar com detalhes de baixo nível, relacionados à interface gráfica. Esse ambiente é considerado aberto porque suporta qualquer estilo de combinação de metodologias, dentre elas: análise estruturada, modelo de entidade-relacionamento, redes de Petri.

Os recursos disponíveis para a construção de uma nova metodologia utilizando as rotinas desse aplicativo são:

- Diversos objetos gráficos com funções pré-definidas de criação, seleção, remoção e mudança de características visuais;
- Objetos conectados logicamente com reorganização automática do diagrama, caso a posição de algum objeto seja alterada;
- Decomposição hierárquica automática dos diagramas, nos quais são organizados em páginas aninhadas de níveis de detalhes;
- Formulários padronizados para obtenção das informações dos objetos gráficos;
- Menus pré-definidos para manipular e construir arquivos e diagramas.

7.4.2. Estilos de diálogos

Existem três fatores que influenciam na escolha do estilo de interação do software a ser construído: o objetivo do software, o tipo de usuário que utilizará o software e os recursos disponíveis para confeccioná-lo [Shneiderman87].

No nosso caso, o objetivo do software é de entrada de dados e processamento. Para este tipo de tarefa o formulário é o diálogo mais apropriado. Através dele muitas informações podem ser entradas, eficientemente, por assemelhar-se ao preenchimento de uma ficha familiar ao usuário.

Como normalmente o usuário da ferramenta FADO será ocasional e pouco experiente, a utilização de menu é aconselhada para o fornecimento das opções de trabalho, pois evita a memorização de teclas de funções.

O terceiro fator influente na escolha dos diálogos é o tipo de recurso disponível no software utilizado para confeccionar a ferramenta. Como aconselhamos utilizar o aplicativo Design/OA, contamos com mais dois estilos de diálogos: diagramas e listas.

7.4.3. Entrada do sistema

Nesta proposta, a entrada do sistema é feita através da tela de abertura mostrada na figura 7.2. Essa tela mostra a identificação do sistema : código e nome do software e a identificação do dicionário utilizado, bem como o nome do usuário e sua senha. Por essa senha controlamos o acesso ao metanível. Após essa tela aparece a tela do menu principal, permitindo iniciar o trabalho.

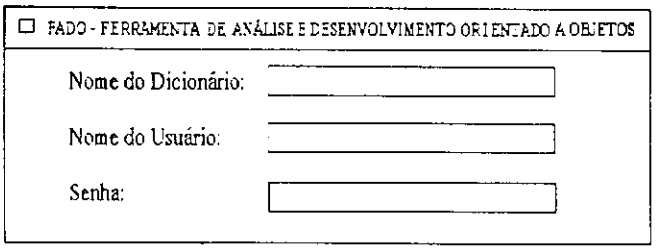

Figura 7.2 - Tela de abertura

7.4.4. Funções da interface exclusivas de FADO

Esta seção mostra as funções da interface que dão suporte, exclusivamente, às técnicas da metodologia FADO. Essas funções tratam de como a interface faz a apresentação, seleção e entrada das informações, como ela fornece a saída das informações e ajuda ao usuário, como as operações são finalizadas e como os erros são corrigidos.

7.4.4.1. Apresentação das informações

As informações são apresentadas de formas distintas em função dos tipos de diálogos utilizados: menus, diagramas e listas.

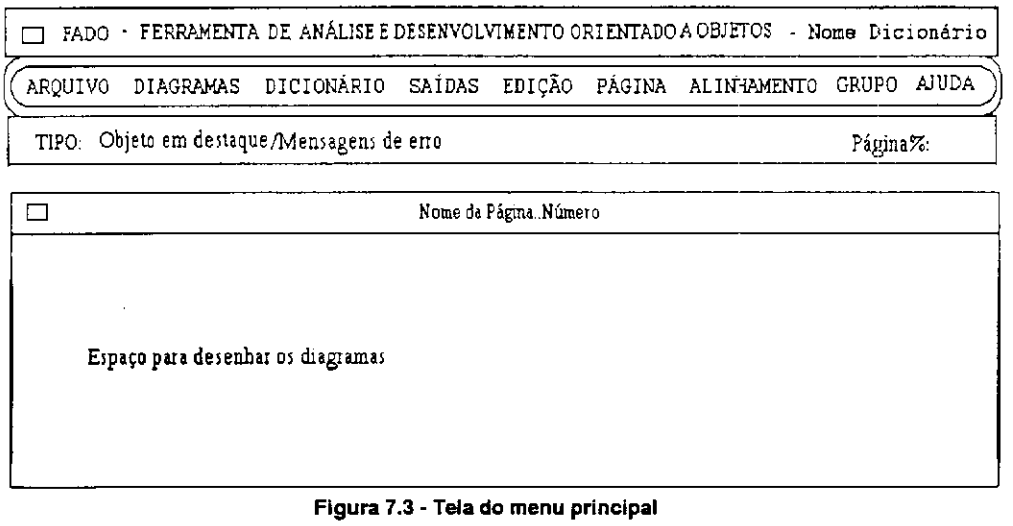

Figura 7.3 - Tela do menu principal

Os menus contêm as opções de trabalho em apenas dois níveis. No menu principal, as funções estão num nível de abstração mais alto e organizadas de acordo com a ordem de frequência da utilização e relevância da opção, conforme vemos na figura 7.3. No menu *pull-down* de cada item do menu principal, há um agrupamento lógico dos itens com um delimitador entre eles por grupo.

Durante a construção dos diagramas, está disponível uma linha de mensagem, com os seguintes campos: tipo e percentual de espaço da página utilizado por um diagrama (ver figura 7.3). Esses campos têm comportamentos dinâmicos, isto é, as informações variam de acordo com o que estivermos fazendo. Por exemplo, no caso da edição de uma figura é mostrado no campo *Tipo*, se essa figura é um conector, label, círculo ou outras combinações, e no campo *Página %*, o percentual ocupado por aquele diagrama na página. 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.

Para maior facilidade de manuseio das páginas, elas são nomeadas e numeradas, e há uma opção para vermos a estrutura das páginas mostrando a hierarquia dos diagramas.

Os diagramas são organizados hierarquicamente, onde cada nível do diagrama está disposto numa única página. Existe a página pai e a página filha, onde esta última é a explosão das idéias genéricas contidas na página pai associada. A figura 7.4 mostra como a interface apresenta os diagramas em níveis de abstração diferentes. No exemplo, consideremos um diagrama estático em que a classe *Companhia* se relaciona com os componentes da classe *Empregado*: *Nome e Endereço*. Se neste nível mais alto de abstração (diagrama da página 1) não interessa o detalhamento de *Empregado*, é aconselhável movê-lo para um diagrama de nível de abstração mais baixo (página 2 que é subpágina de página 1).

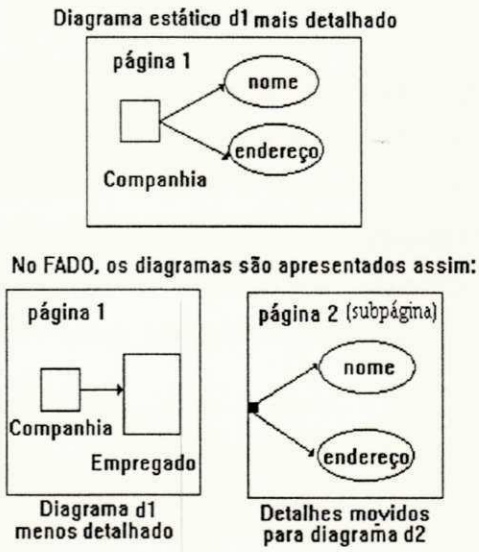

Figura 7.4 - Níveis dos diagramas

As listas são utilizadas para mostrarmos as informações dos elementos do dicionário. As listas podem ser simples ou compostas. As listas simples contêm somente o nome do elemento. As listas compostas contêm mais de uma informação do elemento. Na figura 7.5 vemos uma lista composta pelo número e nome do elemento, pela classe que o elemento pertence e pelo diagrama onde ele aparece. Essa lista é utilizada para consultar todos os métodos do dicionário de dados, que saem da classe *Empregado*.

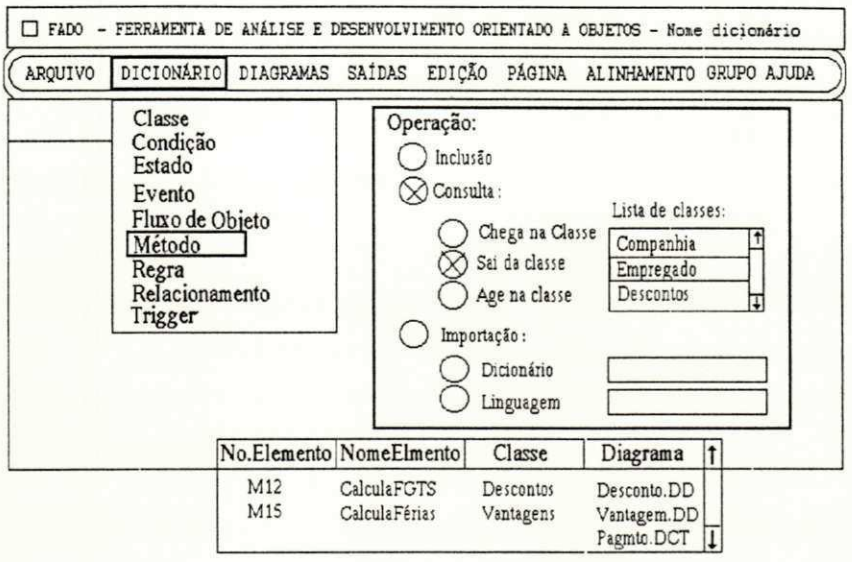

Figura 7.5 - Tela do menu Dicionário

7.4.4.2. Seleção das opcões

Quando trabalhamos com o *mouse*, a seleção das opções pode ser feita diretamente clicando o botão sobre a opção desejada. Por exemplo, no menu de arquivos da figura 7.6, para abrirmos um arquivo, escolhemos uma das opções: diagrama estático, contextual, estado, dinâmico, arquivo de inconstistências ou do esquema lógico TOM. Depois, aparecem na lista de arquivos todos aqueles com a extensão do arquivo conforme a opção escolhida. Com o click do mouse, o usuário seleciona o arquivo para trabalhar. Como esse menu é semelhante ao menu *FILE* fornecido pelo Design/OA, as explicações

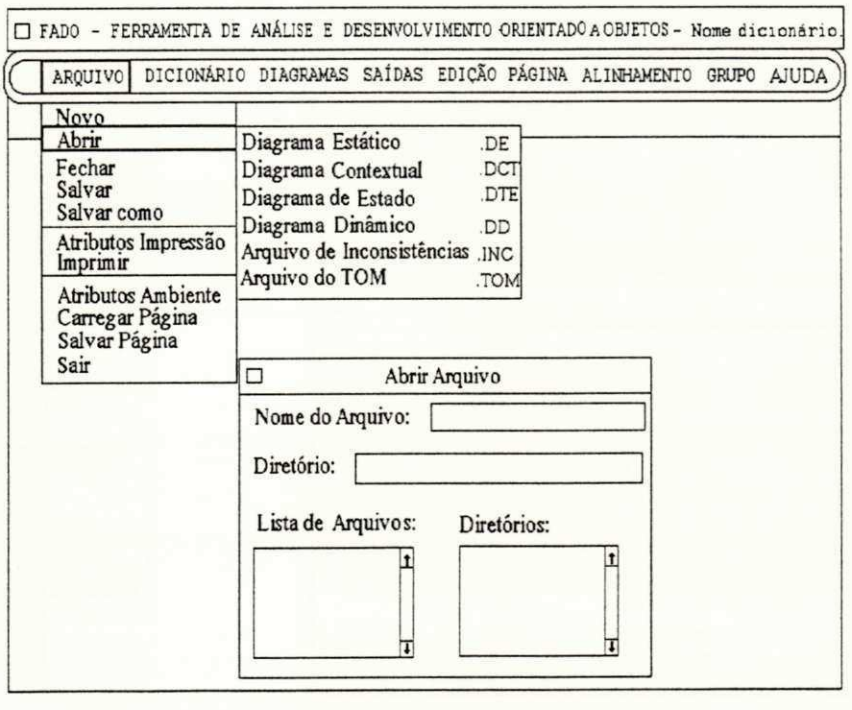

Figura 7.6 - Tela menu Arquivo

das outras opções desse menu podem ser encontradas com detalhes no manual do Design/OA [Metasoft89]. O usuário deve observar que as opções disponíveis estarão em *highlight,* caso contrário, elas dependem do acontecimento de algum evento e/ou condição para que se tornem disponíveis.

7.4.4.3. Entrada das informações

As informações dos elementos a serem colocadas no dicionário de dados devem seguir o mesmo formato dos modelos dos formulários desses elementos. Os modelos padrões desses formulários foram descritos no capítulo 5. A entrada dessas informações no dicionário de dados pode ser feita de duas maneiras: pela importação de classes ou hierarquias de classes de outros projetos ou linguagens orientadas a objetos ou pela digitação direta nos próprios formulários.

No primeiro caso, vimos na figura 7.5 que ao importarmos uma classe devemos informar qual dicionário ou linguagem ela pertence.

No segundo caso, os formulários podem ser chamados diretamente por uma das opções do menu *Dicionário* (vistas na figura 7.5) ou durante a criação dos elementos nos diagramas. Por exemplo, na figura 7.7, depois que desenharmos uma classe num diagrama estático, é aberta uma janela com o formulário de classe, e após preenchermos o formulário, o nome da classe é colocado dentro da figura representativa do elemento e as outras informações vão para o dicionário de dados. Se a classe desenhada já estiver em outro diagrama, mas do mesmo dicionário não é mais necessário preencher as informações novamente do formulário, basta dizer o nome da classe e as informações são trazidas para dentro do formulário.

A seguir mostraremos os elementos utilizados para construírmos os diagramas e os fomulários associados a esses elementos. Cada elemento desenhado possui um ou mais formulários a serem preenchidos após sua criação no diagrama. Esses elementos estão na primeira parte do menu *Diagrama.*

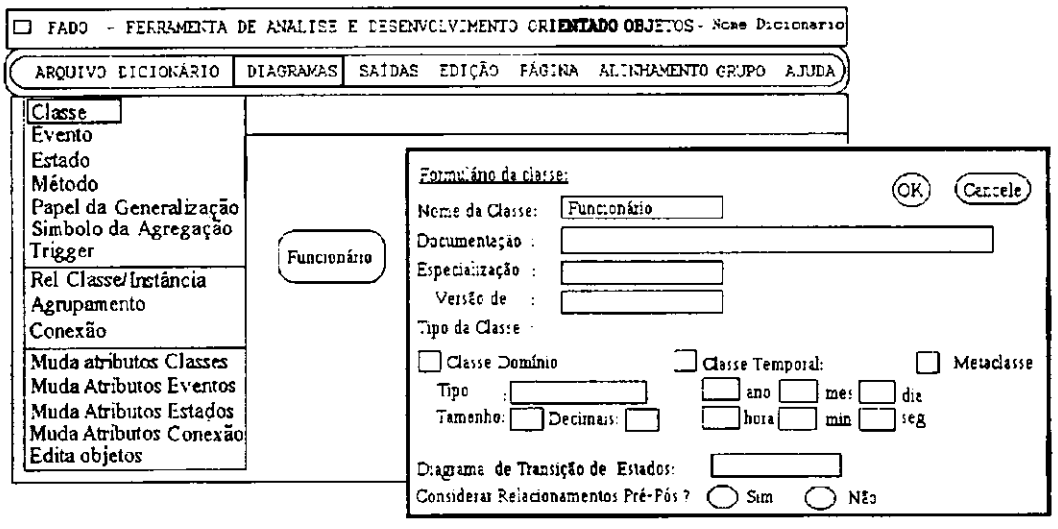

Figura 7.7 - Tela menu Diagramas

- Classe – a figura representativa da classe é desenhada nos diagramas estáticos, dinâmicos e contextual e o formulário associado é de classe.
- Evento – a figura representativa do evento é desenhada no diagrama dinâmico e o formulário utilizado é de evento.
- Estado – quando a figura representativa dos estados de um objeto for desenhada no diagrama de transição de estado haverá um formulário de estado contendo somente o nome do estado e o nome do objeto.
- Método – a figura representativa do método é desenhada no diagrama dinâmico e os formulários que aparecem são de método e o de condição.
- *Trigger* – a figura representativa desse elemento é desenhada no diagrama dinâmico e os formulários associados a figura são de *trigger* e o de condição.
- Papel da generalização – a figura representativa do papel da generalização é desenhada no diagrama estático e após este desenho aparece o formulário de generalização.
- Símbolo da agregação – a figura representativa da agregação é desenhada no diagrama estático e possui o formulário de agregação associado a ela.

As possíveis conexões entre esses elementos estão representadas na segunda parte do menu *Diagramas,* são elas:

- Relacionamentos RC/RI – esta opção é escolhida quando se trata de um relacionamento de classe ou de instância entre duas classes no diagrama estático. O formulário que aparece no momento desta conexão é o formulário de relacionamento RC/RI.
- Agrupamento – esta opção é escolhida quando se trata de fazer a conexão entre a classe grupo e a classe elemento no diagrama estático. O formulário de agrupamento é mostrado neste momento.
- Conexão – esta opção é escolhida quando se trata de uma conexão entre estados (eventos), conexão entre classes no diagrama contextual (métodos), conexão com metaclasses (relacionamento metaclasse) ou conexão no diagrama dinâmico (fluxos de objetos). O formulário que aparece pergunta somente o nome da conexão.

Os atributos das figuras representativas dos elementos de classes, eventos e estados e algumas conexões podem variar quanto ao tamanho, brilhância e outros. Os itens da terceira parte desse menu, permitem escolhermos as variações para esses elementos. A opção *Edita objetos* apresenta na tela o formulário do elemento clicado para alterar suas informações.

7.4.4.4. Saída das informações

Esta opção diz como os resultados processados pelo software são emitidos na interface. Quando se trata de alguma mensagem, esta se posiciona sempre na linha de mensagem. Quando se trata de mostrar um resultado gerado pela ferramenta, por exemplo, o esquema lógico do modelo TOM ou o arquivo de inconsistência, então esses arquivos são mostrados na mesma área livre utilizada para desenhar os diagramas.

As opções das saídas geradas pela ferramenta podem ser vistas na figura 7.8. Quando o usuário termina

de fazer todos os diagramas e de preencher os formulários, ele deve fazer uma análise de consistências antes de gerar o esquema TOM. Antes disso ele deve definir quais regras de validação devem estar ativas durante essa análise. Existe a opção de analisar somente alguns itens para dar maior flexibilidade de trabalho ao usuário. Se após a geração do arquivo de inconsistência nenhum erro for encontrado, o esquema TOM pode ser gerado.

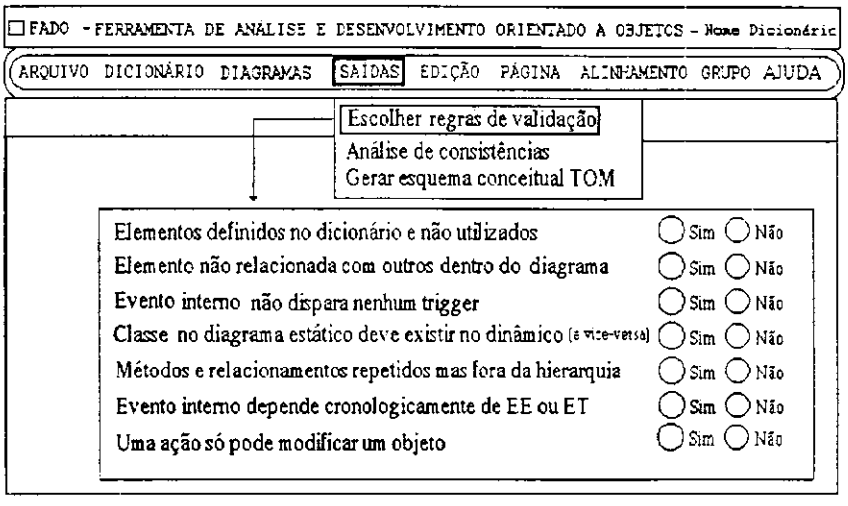

Figura 7.8 - Tela menu Saídas

7.4.4.5. Fornecimento de ajuda on-line

A ferramenta FADO fornece um menu de ajuda em que são explicadas, num formato texto, as fases da metodologia, e no formato gráfico, os passos a serem seguidos para desenhar os diagramas: estático, contextual, dinâmico e de transição de estado. A figura 7.9 mostra as opções desse menu.

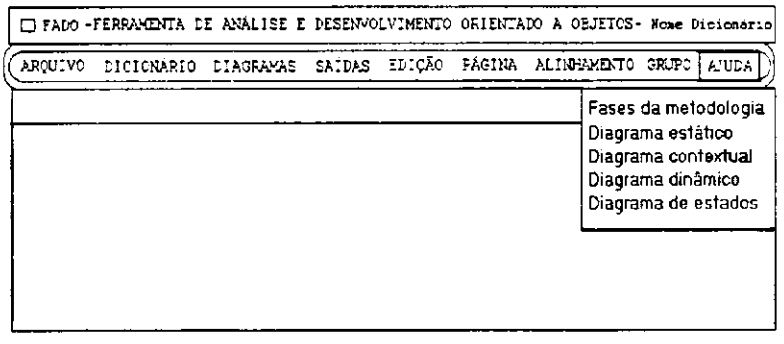

Figura 7.9 - Tela menu Ajuda

7.4.4.6. Orientação no uso da interface

Durante a definição desta proposta de interface, houve uma preocupação em fornecer ao usuário uma orientação constante sobre seu trabalho. Portanto através dos campos *Tipo* e *Nome da página* (figura 7.3), ele sabe onde está e o que faz. Para saber para onde irá, ele dispõe no menu de *Ajuda* a opção *Fases da metodologia*, que diz os passos das fases da metodologia a serem seguidos. Se o usuário quiser saber de onde vem, através da seleção da estrutura de página do menu *Página*, ele pode ver a relação das páginas existentes e a página corrente fica em destaque.

7.4.4.7. Detecção e correção de erros das informações

A ferramenta deve evitar a propagação de erros durante a construção dos diagramas. Por exemplo, não é permitido a conexão entre dois estados com a opção *Relacionamento RC/RI* do menu *Diagrama*, ou a criação do mesmo estado num diagrama de transição de estado, etc.

Mas quando ocorre um erro durante a geração do esquema lógico TOM ou do arquivo de inconsistência, o processamento da geração não é interrompido e o arquivo gerado conterá todos os erros. O usuário deve ler esse arquivo e corrigir as inconsistências. Esse processo deve ser repetido até não existir mais nenhum erro.

Se ocorre um erro no preenchimento do formulário podemos consertar o campo imediatamente ou cancelar a operação. Qualquer atualização no dicionário de dados reflete também nos diagramas e listas.

7.4.4.8. Finalização das operações

Existem casos em que o usuário deseja sair do ponto em que se encontra a qualquer momento. A saída é sempre posicionar o *mouse* no quadradinho situado no canto superior esquerdo e clicar duas vezes. Quando se trata da tela com o menu principal tal execução encerra o serviço, voltando ao sistema operacional.

7.4.5. Funções de apoio da interface FADO

Esta seção mostra as funções disponíveis na interface do aplicativo Design/OA e que achamos ser conveniente incorporá-las na ferramenta FADO. Essas funções são de apoio à construção dos diagramas e estão associadas a 4 menus:

- **Menu Edição** que correponde ao menu *Edit* do Design/OA, através dele podemos selecionar, remover e copiar figuras do próprio diagrama e entre diagramas. As opções são mostradas na figura 7.10, as explicações detalhadas de cada uma podem ser encontradas em [Metasoft89].

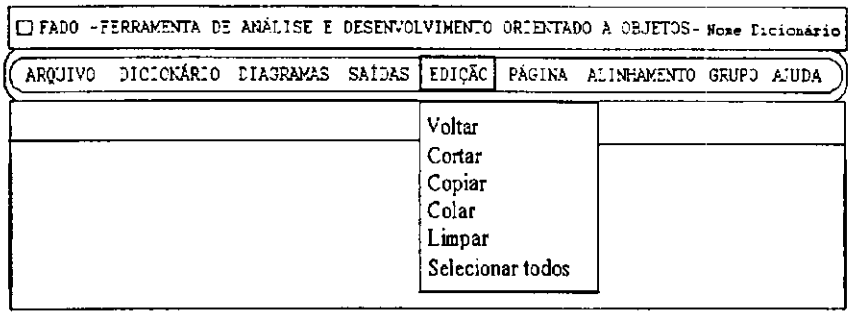

Figura 7.10 - Tela menu Edição

- **Menu Página** o qual é semelhante ao menu *Page* do Design/OA, através dele podemos definir atributos de uma página, conforme vemos na figura 7.11. Uma página pode ser mostrada, aumentada (*Zoom*), diminuída, criada, destruída e inicializada. Para navegarmos entre as páginas devemos escolher os itens página *Pai* e *Filha*. O item *Estrutura*,permite-nos ver todos os relacionamentos hierárquicos entre as páginas. Quando o usuário estiver trabalhando com várias páginas ele poderá se livrar de algumas sem descartá-las totalmente. Feito isto aparecerá um ícone associado àquela página no final da tela. Para trazê-las de volta é só posicionar o *mouse* sobre o ícone e clicar duas vezes.

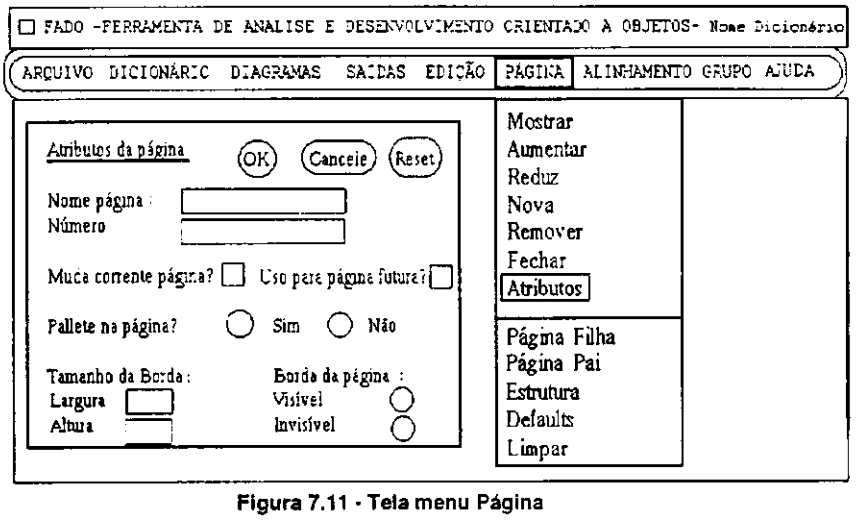

Figura 7.11 - Tela menu Página

- **Menu Alinhamento,** menu que é chamado de *Align* no Design/OA, sua função básica é alinhar uma figura em relação a outra selecionada. Utilizando os itens disponíveis podemos colocar uma figura no centro da página, horizontal ou vertical em relação a outra, ou no meio de outra figura. Existem várias maneiras, mas consideramos estas opções da figura 7.12 as mais úteis.

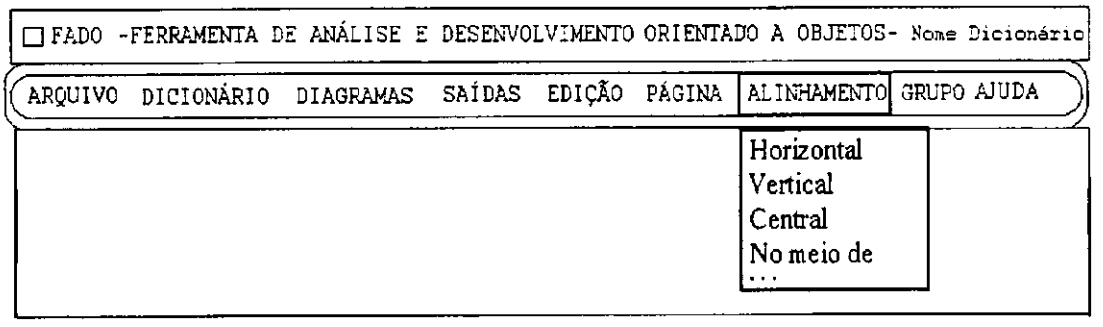

Figura 7.12 - Tela menu Alinhamento

- **Menu Grupo** que corresponde ao *Group* no Design/OA, sua função básica é agrupar figuras para tornar o trabalho mais rápido. Através dos itens mostrados na figura 7.13, podemos formar/desfazer grupos, adicionar/subtrair figuras ao grupo já formado, duplicar o grupo, mover grupos para páginas filhas, trazê-los de volta à página pai e ligar/desligar um elemento com outra página.
  A opção de duplicar um grupo de figuras é equivalente a selecionar um conjunto de figuras com o mouse e depois escolher os itens *Copie* e *Cole* do menu *Edição*. Embora esta função seja semelhante a das utilizadas em *Edição*, ela é de carácter mais contextual , pois trata-se de transportar figuras logicamente agrupadas entre páginas, formando a estrutura hierárquica das páginas, e não apenas deslocamento de posição. Por exemplo, para o diagrama estático da figura 7.4 formamos um grupo dos campos *Nome* e *Endereço* e, em seguida, escolhemos a opção *Explosão*, então esses campos foram automaticamente transferidos para uma nova página, criando a relação pai-filho.

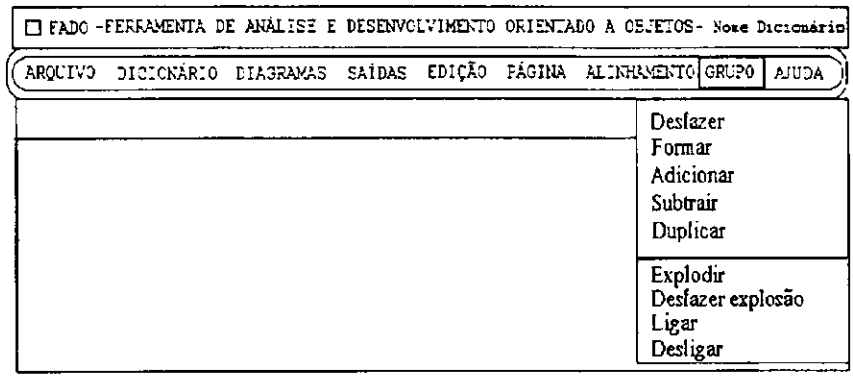

Figura 7.13 - Tela menu Grupo

Capítulo 8

Conclusão

8.1. Considerações finais

Apesar da proliferação de metodologias para apoiar o processo de desenvolvimento orientado a objeto, elas sempre se diferenciam por darem suporte à características distintas. Não existe uma que possa ser usada de forma universal em qualquer situação.

O que sugerimos como alternativa para solucionar o problema da escolha da metodologia mais apropriada para uma determinada aplicação é fazer a avaliação dessa com relação aos critérios de avaliação de qualidade definidos no capítulo 4. Esses critérios apoiam as atividades a serem realizadas durante o processo de desenvolvimento do sistema: construção, avaliação e gerência.

Neste trabalho, fizemos uma avaliação de 5 metodologias orientadas a objetos e descobrimos os seguintes pontos deficientes: suporte à modelagem de características de bancos de dados ativos temporais orientados a objetos, identificação e representação de outros tipos de relacionamentos e abstrações, otimização de classes, representação do comportamento dinâmico (troca de mensagens associadas ao controle de eventos) e integração dos modelos estáticos e dinâmicos através da identidade das classes constantes em todos os diagramas.

A partir destes problemas encontrados e adaptando alguns conceitos das metodologias de Booch para engenharia de software e de Rolland e Schiel para banco de dados orientado a objetos ao ambiente TOM, obtivemos uma ferramenta adequada a problemas complexos de projeto de sistemas de informação: a metodologia FADO.

A metodologia FADO objetiva representar a estrutura e comportamento das classes, através da análise de eventos (controle, sequência e tempo). Houve um tratamento especial para dar suporte ao mecanismo de regras ativas do modelo TOM, onde os aspectos dinâmicos (eventos, *triggers* e condições) foram abordados e representados. Ela também apóia a atividade de gerência de um sistema fornecendo suporte a versões e provém heurísticas para auxiliar na análise e no entendimento do problema.

O processo FADO permite a reusabilidade e iteração, como forma de melhor cobrir à extensibilidade

e manutenibilidade, e a utilização de protótipos, como forma de ir validando o produto durante sua construção. Ambas as estratégias *top-down* e *bottow-up* são utilizadas para especificação do sistema.

A ferramenta FADO permite a importação de classes ou hierarquia de classes de outros projetos já desenvolvidos ou de linguagens de programação orientadas a objetos. As informações importadas devem estar no formato dos modelos dos formulários. Os formulários associados aos elementos dos diagramas, contêm informações, que ao final do processo de desenvolvimento, nos permitem obter o modelo de dados TOM todo especificado.

8.2. Contribuição deste trabalho

Este trabalho inclui os seguintes aspectos relevantes:

- Algumas considerações sobre os conceitos de orientação a objetos e o papel da abstração na modelagem conceitual;
- Uma discussão sobre as funcionalidades dos bancos de dados orientados a objetos;
- Um estudo comparativo entre cinco metodologias de desenvolvimento orientado a objetos, fazendo uma análise crítica das suas características;
- A elaboração de uma metodologia de análise e projeto de banco de dados temporal orientada a objetos;
- Definição de heurísticas que servem como normas de qualidade para auxiliar na construção da representação, dando suporte ao processo de verificação do produto.
- A validação da metodologia com o estudo de um caso real , acompanhando todos os passos do processo FADO, até a geração do esquema conceitual TOM;
- Um estudo do software DesignOA para fazer a definição da interface da ferramenta FADO seguindo as vantagens oferecidas pelo software.

8.3. Trabalhos futuros

Nós apresentamos um projeto de interface da ferramenta FADO, e fizemos a implementação de um programa que cria formulários, simulando um gerenciador de diálogo do DesignOA. O programador cria os formulários utilizando o software GUIDE das estações SUN, e depois executa esse programa para gerar os formulários de acordo com a especificações do software DesignOA. Com isto se evita a contagem de pontos gráficos da tela e se facilita a manutenção das telas. Mas o trabalho de implementação da ferramenta ainda precisa ser feito. Embora o projeto tenha sido feito utilizando os recursos que o software DesignOA oferece nada impede que ele seja implementado com outro software. Como a ferramenta FADO

constrói o projeto conceitual dentro do ambiente DYNAMO, qualquer mudança que ocorra nesse ambiente deve ser suportada pelas características dessa ferramenta. Por exemplo, está previsto a definição de um integrador de visões dos esquemas externos, é possível que ao defini-lo a metodologia FADO terá que suportar um mecanismo de visões. A metodologia FADO pode ser enriquecida se houver o desenvolvimento de um mecanismo de concorrência e um suporte à gerência quanto ao acompanhamento de custos e cronograma.

Apêndice A

Esquema de dados TOM

Neste apêndice ilustramos a modelagem da aplicação estudada usando a linguagem TOM. A seguir, mostramos a notação utilizada nesta descrição e o seu significado:

- (<NomeCampo>) - é a instância da classe de valor igual ao contido na variável NomeCampo;
- Self - se refere a instância corrente;
- ![<classe><método>] - significa que o método dessa classe foi executado para todas as instâncias;
- ![<NomeTrigger>] - significa que o trigger foi disparado;
- {[campo1, campo2]} - significa um conjunto de instâncias, em que cada instância é formada pelos campos campo1 e campo2.

```
classTabelaExemplo
        specialization of G_CLASS
        instance relationships
              NomeTab: Nome (1,1)
              QtdExTab : Quantidade (1,*)
       Identification Key is NomeTab
       grouping of Exemplo explict, parameters: covering
       class methods
              IniciaTe (ntab:NomeTab)
               pre_conditions
                      not (ntab) in TabelaExemplo
               post_conditions
                      (ntab) in TabelaExemplo
                      (ntab).QtdExTab =  0

       instance  methods
             AssDessTE (op:Caracter,ex:Exemplo)
               pre_conditions
                       if op =  "I" then ex elem_of self else not ex elem_of self
                post_conditions
                       if op =  "I" then not ex elem_of self else ex elem_of self
            ReorganizaExs (op:Caracter,tp:Nome,nom:Nome,nov:Nome)
               pre_conditions
                      not self.QtdExTab =  0
               post_conditions
                      if (ex elem_of self) then ![ex ReorganizaEx]
/* enquanto existiram exemplos, foi executado o método ReorganizaEx in Exemplo */
            ConsultaExs
               pre_conditions
                     self. QtdExTab  0
               post_conditions
                      if (ex elem_of self) then ![ex ConsultaEx]
```

```
        SalvaVersão (ntab:NomeTab)
            pre_conditions
                self in TabelaExemplo
            post_conditions
                self.ProximaVersão = new

class Exemplo
    class relationships
        MaxAtrEx: Quantidade (1,1)
    instance relationships
        CodEx: Código (1,1)
    Identification Key is CodEx
    Agregation of {[Atributo,Valor]},ElementoClassificador explicit
    class methods
        IncluiEx (ec:Código, {[na,va]:[Atributo,Valor]} , ne:ElemClassif)
            pre_conditions
                not (ec) in Exemplo
                not {na} = "" and not ne = ""
            post_conditions
                (ec) in Exemplo
                (ec) = [{[na, va]}, ne]
    instance methods
        ExcluiEx
            pre_conditions
                self in Exemplo
            post_conditions
                not self in Exemplo
        ReorganizaEx (op:Caracter,tp:String,nom:String,nov:String)
            pre_conditions
                (tp = "AT" and self.NomeAtr = nom) or
                (tp = "ElCl" and self.NomeElCl = nom)
            post_conditions
                if (op = "E" and tp= "AT") then not [nom,v] elem_of (part_of(Exemplo)) or
                if (op = "A" and tp= "AT") then [nom,v] elem_of (part_of(Exemplo)) or
                if (op = "A" and tp= "ElCl" ) then not nom part_of (Exemplo)
        ConsultaEx
            pre_conditions
                self in Exemplo

class Atributo
    instance relationships
        TipoAtr: TipoAtributo (1,1)
        NomeAtr: Nome (1,1)
        TemValor: Valor (0,*)
    Identification Key is NomeAtr
    class methods
        IncluiAtr (na:Nome, tp:TipoAtributo, vl:Valor)
            pre_conditions
                not (na) in Atributo
            post_conditions
                (na) in Atributo
                (na).TipoAtr = tp
                (na).TemValor = vl
    instance methods
        ExcluiAtr
```

```
            pre_conditions
                self in Atributo
            post_conditions
                not self in Atributo
        ConsultaAtr
            pre_conditions
                self in Atributo
        AlteraAtr
            pre_conditions
                self in Atributo
    Parameters with time 00:06:00:00:00:00


class ElemClassif
    instance relationships
        NomeElCl: Nome (1,1)
    Identification Key is NomeElCl
    class methods
        IncluiElCl(ne:NomeElCl)
            pre_conditions
                not (ne) in ElemClassif
            post_conditions
                (ne) in ElemClassif
    instance methods
        ExcluiElCl
            pre_conditions
                (ne) in ElemClassif
            post_conditions
                not (ne) in ElemClassif
        ConsultaElCl
            pre_conditions
                self in ElemClassif

class TabelaFreqAtr
    instance relationships
        TemTE : TabelaExemplo (1,1)
    Identification Key is TemNome
    Agregation of {[Atributo,Frequencia]},explicit
    class methods
        IncluiFreq(nf:TabelaFreqAtr,nt:TabelaExemplo ,[na,TemFq]:[Atributo,Frequencia])
            pre_conditions
                not na = " " and not nt = " " TemFq < 3 otherwise
                warning ("Atr:",na,"pouco frequente") /* se a frequencia do atributo é menor 3
dar aviso*/
            post_conditions
                (nf).TemTE = nt
                (nf) = [na, TemFq]
    instance methods
        ConsultaFreq (nt:TabelaExemplo,na:Atributo)
            pre_conditions
                self.TemTE = nt
                (na) is (part_of (TabelaFreqAtr))
 /* o atributo deve ser fazer parte do conjunto de atributos da TabelaExemplo dentro de uma
TabelaFreqAtr */


class Gerente
```

```
    instance relationships
        TemConfig: ConfigAmb(1,1)
        TemEntrev: OpEntrev (1,1)
        QtdExDig : Quantidade (1,1)
        TempoDig : Integer (1,1)
        TemVersãoAuto: VersãoAutomática (1,1) inverse of EVersãoDe with 20 old values
    Identification Key is TemConfig
    Agregation of { [Algoritmo,TabelaExemplo,MatrizRelevância]} explicit
    class methods
        ConfiguraAmb (TemConfig:ConfigAmb, te,qt,td,tva:integer)
            pre_conditions
                not td = 0 and not qt = 0
            post_conditions
                (TemConfig) in Gerente
                (TemConfig).TemEntrev = te
                (TemConfig).TemVersãoAuto = tva
                (TemConfig).QtdExDig = qt
                (TemConfig).TempoDig = td
    instance methods
        AssAlgMRTE ({[al,te,mr]: [Algoritmo,TabelaExemplo,MatrizRelevancia] })
            pre_conditions
                not na = " " and not ne = " "
            post_conditions
                self = [al, te, mr]
        GerenciaExAlg (na:NomeAlg,ab:AceitaBuraco,av:AceitaVlCont)
            pre_conditions
                self.NomeAlg= na
            post_conditions
                if ab = "N" then ![T7] /* se o algoritmo não aceitava buraco foi disparado
                        trigger T7 */ if av = "N" then ![T8]
        GerenciaAlim (nt:TabelaExemplo,ex:Exemplo)
            pre_conditions
                (ex) in TabelaExemplo
            post_conditions
                if (self.TemVersãoAuto = "S" and self.QtdExDig = soma) then ![T4]
                /* a cada QtdExDig e se a opção por guardar versão automática é ativa, o
trigger T4 deve ter sido disparado */
                if av = "N" then ![T8]

class Algoritmo
    instance relationships
        NomeAlg : Nome (1,1)
        TemBuraco : AceitaBuraco (1,1)
        TemVlCont : AceitaVlCont(1,1)
        TemCorpo : Corpo(1,1)
    Identification Key is NomeAlg
    class methods
        IncluiAlg ( na:NomeAlg,tb:AceitaBuraco ,tvc:AceitaVlCont,Tc:corpo)
            pre_conditions
                not (na) in Algoritmo
            post_conditions
                (na) in Algoritmo
                (na).TemBuraco = tb
                (na).TemVlCont = tvc
                (na).TemCorpo = tc
    instance methods
        ExcluiAlg
```

```
            pre_conditions
                    self in Algoritmo
            post_conditions
                    not self in Algoritmo
        ConsultaAlg
            pre_conditions
                    self in Algoritmo
        ExecutaAlg
            pre_conditions
                    self in Algoritmo

class Relevância
    instance relationships
        CodRel : Codigo (1,1)
    Identification Key is CodRel
    Agregation of ElemClassif, Atributo, Valor, Pertinência explicit
    inheritance of methods (IncluiRelev, computada (IncluiElCl,IncluiAtr,IncluiPert))
    class methods
        IncluiRelev (ta:TabelaExemplo, rel:Relevância, [ne,na,va,p]:
                    [ElemClassif, Atributo, Valor, Pertinência])
            pre_conditions
                    not (rel) in Relevância
                    [na,va] elem-of (part-of(ta))
              /* o par atributo/valor já deve existir na TabEx */
            post_conditions
                    (rel) in Relevância
                    (rel) = [ne,na,va,p]
    instance methods
        ExcluiAlg
            pre_conditions
                    self in Algoritmo
            post_conditions
                    not self in Algoritmo
        ConsultaAlg
            pre_conditions
                    self in Algoritmo
        ExecutaAlg
            pre_conditions
                    self in Algoritmo
-

class MatrizRelevância
    instance relationships
        NomeMR: Nome (1,1)
        Codrel : Codigo (1,*)
    Identification Key is NomeMR
    grouping of Relevância explict, parameters: covering
    instance methods
        AssDessMR (op:Caracter,rel:Relevância)
          pre_conditions
                if op = "I" then rel elem_of self else not rel elem_of self
          post_conditions
                if op = "I" then not rel elem_of self else rel elem_of self
      ConsultaRels
          pre_conditions
                self. QtdExTab 0
          post_conditions
```

```
                if (rel elem_of self) then ![rel ConsultaRelev]

class Pertinência
    instance relationships
        TemPert : Valor (1,1)
    Identification Key is TemPert
    class methods
        IncluiPert (p:Pertinência)
            pre_conditions
                not (p) in Pertinência
            post_conditions
                (p) in Pertinência

class Entrevistador
    instance relationships
        LinhaMens : String (1,*)
    class methods
        SugereRelev (nt:TabelaExemplo,mr:MatrizRelevância)
            pre_conditions
                (nt) in TabelaExemplo
            post_conditions
                ∀ (nt).NomeElCl part-of (mr)
/* todos os ElemClassif da TabelaExemplo fazem parte de alguma MatrizRelevância */
        MostraCritica (linha:LinhaMens)
            pre_conditions
                (linha) not = " "

class Otimizador
    instance relationships
        TemFq: Frequência (1,1)
        QtdAmb: Integer (1,1)
    class methods
        TrataFreq (nt:TabelaExemplo)
            pre_conditions
                (nt) in TabelaExemplo
            post_conditions
                (nt) in TabelaFreqAtr
                ¬∀ (nt).NomeAtr part-of TabelaFreqAtr
            /* todos os Atributos da TabelaExemplo estão na TabelaFreqAtr */
        TrataAmb (nt:TabelaExemplo)
            pre_conditions
                (nt) in TabelaExemplo
            post_conditions
                QtdAmb = 0 otherwise warning
                "Tabela Exemplo contém:", QtdAmb,"exemplos ambíguos"
        TrataBuraco (nt:TabelaExemplo, nf:TabelaFreqAtr)
            pre_conditions
                (nt) in TabelaExemplo
                (nf).TemTE = nt
            post_conditions
                ¬∀not ( (nf).Frequencia = 0 )
        /* todos os Atributos da TabelaFreqAtr devem ter frequência diferente de 0 */
        TrataDiscr (na:Atributo)
            pre_conditions
                (na).TipoAtr = "C"
            post_conditions
```

```
class Frequência
    type Integer

metaclass G_Class
        specialization of TOM_CLASS
        instance relationships
            TemVersão V_Class (1,*) inverse of VersãoDe
            Identificação: O_ID (1,1)
        class methods
            NewVersionedObject
                pre_conditions
                    not V in self V in O_ID
                post_conditions
                    V in self

metaclass V_Class
        specialization of HistoryObject
        instance relationships
            ProximaVersão: V_Class (1,*)
            PreviaVersão: V_Class (0,*)
            VersãoDe: G_Class (1,1) inverse of TemVersão
        instance methods
            alternative_of
              pre_conditions
                  # (PreviaVersão(self)) = 1
              post_conditions
                  let v be [PreviaVersão (self) next]
                  historyBegin[v] = historyEnd[self]
            part_of_v
              pre_conditions #
                  (PreviaVersão(self)) = 1
              post_conditions
                  let vo be [PreviaVersão (self)]
                  let v be [vo next]
                  contained (Svo,union (Sself,Sv))
                  intersec(Sself,Sv) = { }
                  historyEnd[self] = historyEnd[v]


metaclass Regra
    instance description:
   /* para as proximas metaclasses mostraremos os relacionamentos instanciados */
        instance relationships
            has_name: R1
            has_events: ExemploIncluído
            has_trigger: T1,T2
        instance relationships
            has_name: R2
            has_events: Reorganização
            has_trigger: T3
        instance relationships
            has_name: R3
            has_events: TratFreqAmb
            has_trigger: T4,T5
        instance relationships
```

has_trigger: T6
instance relationships
has_name: R5
has_events: TratDiscrBuraco
has_trigger: T7,T8

metaclass Evento_BD
specialization Evento
instance description:
instance relationships
has_name: ExemploIncluído
active: sim
of_rule: R1
has_method: IncluiEx
has_class: Exemplo
instance relationships
has_name: Reorganização
active: sim
of_rule: R2
has_method: ReorganizaExs
has_class: TabelaExemplo
instance relationships
has_name: TratFreqAmb
active: sim
of_rule: R3
has_method: GerenciaAlim
has_class: Gerente
instance relationships
has_name: TratDiscrBuraco
active:sim
of_rule: R5
has_method:GerenciaExAlg
has_class: Gerente

metaclass Evento_Temporal_Relativo
specialization Evento_Temporal
instance description:
instance relationships
has_name: TempoDigitação
active: sim
of_rule: R4
when: 00:00:00:00:30:00
delay:00:00:00:00:00:00
type: contínuo
event: ExemploIncluído

metaclass Trigger
instance description:
instance relationships
has_name: T1
status: sim
priority: 2
seq_exec:antes
condition: C1
name_action: ConsultaAtr,ConsultaElcl,AssDessTE
instance relationships

has_name: T2
status: sim
priority: 1
seq_exec: antes
condition: C2
name_action: GerenciaAlim
instance relationships
has_name: T3
status: sim
priority: 1
seq_exec: antes
condition: C3
name_action: ExcluiEx
name_faction: ReorganizaEx
instance relationships
has_name: T4
status: sim
priority: 4
seq_exec:antes
condition: C4
name_action: SalvaVersão
instance relationships
has_name: T5
status: sim
priority: 1
seq_exec:antes
condition: C5
name_action: TrataFreq, TrataAmb
instance relationships
has_name: T6
status: sim
priority: 1
seq_exec:antes
condition: C6
name_action: SugereExEl
instance relationships
has_name: T7
status: sim
priority: 2
seq_exec:antes
condition: C7
name_action: SalvaVersão, TrataBuraco
instance relationships
has_name: T8
status: sim
priority: 1
seq_exec:antes
condition: C8
name_action: SalvaVersão, TrataDiscr

metaclass Condição
instance description:
instance relationships
has_name: C1
predicate: (na) in Atributo and (ne) in ElemClassif
otherwise: cancele
has_message: "Atributos e ElCl devem ser primeiro incluídos"

```
instance relationships
        has_name: C2
        predicate: (ex) in Exemplo
        otherwise: Cancele
        has_message: "Exemplo não incluído"
instance relationships
        has_name: C3
        predicate: tp = "EICI" and op = "E"
instance relationships
        has_name: C4
        predicate: self.TemVersãoAuto = "S" and self.QtdExDig = soma
instance relationships
        has_name: C5
        predicate: self.OpEntrev = "S"
instance relationships
        has_name: C6
        predicate: self.OpEntrev = "S"
instance relationships
        has_name: C7
        predicate: (na).AceitaBuraco = "N"
        otherwise: Cancele
        has_message: "Este algoritmo não pode aceitar buraco"
instance relationships
        has_name: C8
        predicate: (na).AceitaVlCont = "N"
        otherwise: Cancele
        has_message: "Este algoritmo não pode aceitar valor contínuo"
```

Referências bibliográficas

[Bailin89] Bailin, S.C. "An Object-Oriented Requirements Specification Model". Communications of the ACM. Vol. 32, no. 5, pp: 608-623, Maio 1989.

[Bancilhon88] Bancilhon F., "Object-Oriented Database Systems". Rapport Technique Altair. Pp:16-88.1988.

[Bancilhon89] Bancilhon F., Delobel C., Kannelakis P., "The O2 Book". Copyright Altair. 1989.

[Baptista92] Baptista A., "Uma Metodologia de Análise e Projeto de Sistema Orientada a Objetos". Dissertação de mestrado. UFPE Departamento de Informática. 1992.

[Boehm4] Boehm B.W., "Verifying and Validating Software Requirements and Design Specifications", IEEE Software, janeiro 1984.

[Booch83] Booch G., "Object Oriented Design". IEEE Software, fevereiro 1996.

[Booch91] Booch G., "Object Oriented Design with Applications". Benjamin/Cummings. Redwood City. 1991.

[Brand88] Brand N., Brinkkemper S., Moormann J., "Deterministic Modelling Procedures for Automatic Analysis and Design Tools". Computerized Assistence During the Information Systems, North Holland. Pp: 117-130. 1988.

[Bretl89] Bretl N., Maier D., Otis A., Penney J., Schuchardt B., Stein J., Williams E. e Willians M., "The GemStone Data Management System". Em Object-Oriented Concepts, Databases and Applications. Kim W., Lochovsky F. ACM Press. Pp:283-308. 1989.

[Capretz92] Capretz L.F., Lee P., "A Classification of Object-Oriented Development Methodologies". ACM SIGPLAN Notices. Vol 27, No.4. Abril 1992.

[Carvalho93] Carvalho A., "Tom Rules: Um monitor de Eventos, Regras e Gatilhos em um Ambiente Orientados a Objetos". Dissertação de mestrado. UFPB/COPIN. 1993.

[Chen76] Chen P., "The Entity-Relationship Model - Toward a Unified View of Data". ACM Transactions on Database Systems. Março 1976.

[Cirne91] Cirne, W.: O Uso de Semântica na Melhoria de Métodos Indutivos de Aquisição Automática de Conhecimento. Dissertação de mestrado. UFPB/COPIN. 1992.

[Crispim91] Crispim E., "Avaliação de Métodos para o Desenvolvimento de Softwares". Dissertação de mestrado. COPPE Sistemas/UFRJ. 1991.

[David92] David M., "Descrição Formal da Estrutura do Modelo Orientado a Objetos Temporal - TOM". Dissertação de Mestrado. UFPB/COPIN. 1992.

[Falquet88] Falquet G., Guyout J., Junet M., Leonard M., Bursens R., Crausaz P., Prince I., "Concept Integration as an Approach to Information Systems Design". Computerized Assistence During the Information Systems Life Cycle. North Holland. Pp: 19-75. 1988.

[Fishman89] Fishman D.H., Annevelink J., Chow E., Connors T., "Overview of Iris DBMS". em Object-Oriented Concepts, Databases and Applications.Kim W., Lochovsky F. ACM Press.Pp:219-250.1989.

[Furtado93] Furtado E.,Schiel U., "Uma Metodologia para Projeto de Banco de Dados Temporal Orientado a Objetos". VIII Simpósio Brasileiro de Banco de Dados. Campina Grande. 1993.

[Guck88] Guck R., Jagnnathan D., Fritchman B., "SIM: A Database System Based on The Semantic Data Model" ACM Sigmod. 1988.

[Heitz87] Heitz M., "HOOD: Hierarchical Object-Oriented Design for Development of Large Technical and Realtime Software" CISI Ingenierie. Novembro 1987.

[Henderson90] Henderson S. Edwards J., "Oriented Systems Life Cycle". Communications of the ACM. Setembro 1990.

[Jackson75] Jackson M.A., "System Development". Prentice Hall. 1975.

[Kerth88] Kerth N., "MOOD: A Methodology for Structured Analysis into Object-Oriented Design". Tutorial OOPSLA'88. San Diego 1988.

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

[Kim89] Kim W., Ballou N., Chou H., Garza J., Woelk D., "Features of the ORION Object-Oriented DataBase System". Em Object-Oriented Concepts, Databases and Applications. Kim W., Lochovsky F. ACM Press. Pp:251-282. 1989.

[Kim90] Kim W., "Object-Oriented Databases: Definition on Research Directions". IEEE Transaction on Software Engineering. Vol 2. No. 3. Setembro 1990.

[Lohman91] Lohman G., Lindsay B., Pirahesh H., "The Starburst Data Model". Communications of the ACM. Outubro. 1991.

[Mark85] Mark L., Roussopoulos N., "The New Database Architecture Framework – A Progress Report". Information Systems: Theoretical and Formal Aspects. North-Holland. 1985.

[Mattoso89] Mattoso A., "Taba Obj: Um Ambiente de Desenvolvimento de Software com Orientação a Objetos". Dissertação de mestrado. COPPE Sistemas/UFRJ. 1989.

[McLeod85] McLeod D., "Abstraction in Databases". Proceedings of the Workshop on Data Abstraction. Pringree Park. 1981.

[Metasoft89] Metasoftware Corporation "Design/OA – Developer's Manual". 1989.

[Meyer88] Meyer B., "Object-Oriented Software Construction". Prentice Hall. 1988.

[Monarchl92] Monarchl D., Puhr G., "A Research Typology for Object-Oriented Analysis and Design". Communications of the ACM. vol.35,no.9. Setembro 1992.

[Mongiovi90] Mongiovi G. ,Cirne W., "Uma Algoritmo Baseado em Conhecimento Para Ampliar a Pontencialidade do ID3. Anais do VI Simpósio Brasileiro de Inteligência Artificial. Campina Grande. 1990.

[Monte88] Monte L.C., "A invasão dos objetos". Trabalho de pesquisa. COPPE – Sistemas/UFRJ. 1988.

[Navathe92] Navathe S., "Evolution of Data Modeling for Databases". Communications of the ACM. Setembro 1992.

[Norman92] Norman Y., "Two Models of Object-Oriented Programming and The Common Lisp Object System". ACM SIGPLAN Notices. Vol 27, No.4. Abril 1992.

[Rolland88] Rolland C., Cauvet C., Proix C., "The Rubis System". CRIS88. Computerized Assistance during the Information System Life TCycle. North Holland. Setembro 1988.

[Rolland89] Rolland C., Cauvet C., Proix C., "A Design Methodology for Object-Oriented Database". In Proc. Management of Data, Ed. Naveen Prakash. Pp:39-65. 1989.

[Rolland90] Rolland C., Souveyet C., "Correction of Conceptual Schemas" . Em Advanced Information Systems Engineering. Springer-Verlag. Pp: 152-174. 1990.

[Rolland91] Rolland C., Cauvet C., "Modelisation Conceptuelle Orientee Objet". VII émes Journées Bases de Données Avancées, Lyon. 1991.

[Rumbaugh91] Rumbaugh J.E., "Object-Oriented Modelling and Design". Prentice Hall. 1991.

[Schasching92] Schaschinger H., "ESA – An Expert Supported OOA Method and Tool". ACM Software Engineering Notes. Vol 17. No.2. Abril 1992.

[Schiel89] Schiel U. "Vodak Version Model (VVM)", Working Paper, GMD/IPSI, Darmstadt/Germany,1989.

[Schiel90] Schiel U., Mistrik I., "Using Object-Oriented Analysis and Design for Integrated Systems ". Procedure International Conference on Systems Integration, IESI, New Yersey. 1990.

[Schiel91] Schiel U.,"An Open Environment for Objects with Time and Versioning". Procedure EastEuroope, Bratislava. 1991.

[Schiel92] Schiel U. et al, "Ambiente Aberto para Desenvolvimento de Banco de Dados Ativos Distribuídos", Relatório PROTEM – Universidade Federal da Paraíba, Campina Grande. 1992.

[Schiel93] Schiel U., Carvalho F. "Tom-Rules: A Uniform And Flexible Approach to Events, Constraints and Implicit Information". Anais VIII Simpósio Brasileiro de Banco de Dados. Campina Grande. 1993.

[Shneiderma87] Shneiderman B., "Designing The User Interface". Addison Wesley. 1987.

[Shaw84] Shaw M., Mellon C., "Abstraction Techniques in Modern Programming Languages". IEEE Software. Pp: 10-26. Outubro 1984.

[Shipman81] Shipman J., "The Funcional Data Model and The Data Language DAPLEX". ACM Transactions on Database Systems. Vol.6, no. 1. 1981.

[Shlaer90] Shlaer S., Mellor S., "Análise de Sistemas Orientada para Objetos". Mc-Graw-Hill. 1990.

[Stonebraker91] Stonebraker M., Kemnitz G., "The POSTGRES Data Model and Query Language". Communications of the ACM. Outubro. 1991.

[Takahashi90] Takahashi T., Liesenberg H., Xavier D., "Programação Orientada a Objetos". VII Escola de Computação. São Paulo 1990.

[Vasco93] Vasco J.J., "Um Ambiente de Apoio à Aquisição Automática de Conhecimento". Dissertação de mestrado. UFPB/COPIN. 1993.

[Yourdon92] Coad P., Yourdon E., "Análise Baseada em Objetos". Campus. 1992.

[Warnier77] Warnier J., Orr K., "Structured Systems Development". Yourdon Press. 1977.

[Winblad93] Winblad A., Edwards S., King D., "Software Orientado ao Objeto". Makron Books. 1993.