Universidade Federal do Rio Grande do Sul
Instituto de Informática
Curso de Pós-Graduação em Ciência da Computação

Estudo Comparativo de Técnicas de Análise Orientada a Objetos

por

Guillermo Bustos Reinoso

T.I. n° 317 CPGCC-UFGRS Março 93

Trabalho Individual I

Prof. Carlos A. Heuser
Orientador

Porto Alegre, março de 1993.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O trabalho apresenta uma definição da análise orientada a objetos e uma descrição dos aspectos da modelagem deste tipo de análise: aspecto estrutural, aspecto dinâmico e aspecto funcional.

É realizado um estudo das diversas técnicas de análise sob o paradigma da orientação a objetos, categorizando-as em textuais, evolutivas, integracionistas, reversas e comportamentais, e descrevendo a técnica mais representativa de cada uma destas categorias.

São definidos critérios de comparação e fornecidas tabelas que mostram as características de cada técnica de acordo com estes critérios. Estes critérios são divididos em aqueles relativos ao processo de análise propriamente tal, e aqueles relativos ao resultado deste processo, isto é, à especificação orientada a objetos.

De acordo com a descrição e a comparação das técnicas, são indicados o ponto forte e os pontos fracos que elas apresentam. Como ponto forte é indicado a modelagem estrutural de objetos. Como pontos fracos são indicados o particionamento da complexidade, a reusabilidade, a modelagem funcional encapsulada, a validação dos modelos por parte dos usuários, e a estimação e dimensionamento de sistemas, entre outros.

Palavras-chave: *desenvolvimento orientado a objetos*, *análise de sistemas*, *modelagem orientada a objetos*, *especificação de requisitos*.

ABSTRACT

This work presents an object-oriented analysis definition and a modeling aspects description for this kind of analysis: structural, dynamic and functional aspects.

Differents analysis techniques are studied from the object-oriented paradigm point of view. The techniques are classified in textuals, evolutionals, combinatives, reverses and behaviorals. The most representative technique for each category is described.

Comparison criteria are defined and tables are used to show the techniques characteristics through these criteria. The criteria are divided in those that deal with the analysis process and with the analysis outcomes, i.e. the object-oriented specification.

Object-oriented analysis strength and weaknesses are indicated using the techniques descriptions and comparison. The strength is the object structure modeling. The weaknesses are: complexity partitioning, reusability, encapsulated functional modeling, user validation and size estimation of object-oriented systems.

Key-words: *object-oriented development, systems analysis, object-oriented modeling, requirements specification*.

INTRODUÇÃO

A orientação a objetos pode ser considerada hoje como uma revolução na forma de construção do software. Esta revolução originou-se novamente (o mesmo aconteceu com a denominada revolução *estruturada*) a partir de um paradigma de programação surgido há mais 25 anos atrás com a linguagem Simula 67, que já introduzia os conceitos de *classe* e *herança*. No decorrer do tempo foram surgindo diferentes linguagens orientadas a objetos que demandaram métodos e técnicas para projetar arquiteturas de software e construir programas com orientação a objetos. Com isto surgiu o projeto orientado a objetos, e este, por sua vez, resultou no que atualmente é conhecido como a análise orientada a objetos, que busca conceber o domínio do problema em termos de objetos.

Nos últimos anos, e mais especificamente a partir de 1989, tem surgido uma série de propostas que visam abordar a fase de análise no desenvolvimento de software com orientação a objetos. Assim hoje é possível contabilizar mais de 20 propostas diferentes de técnicas, em cujo título constam as palavras chaves de *Análise* e *Orientação a Objetos* ou alguns dos seus sinônimos, em artigos em revistas especializadas, trabalhos apresentados em congressos e livros técnicos.

O propósito deste trabalho é apresentar um estudo comparativo de algumas destas técnicas de análise orientada a objetos. Para tal efeito devem ser atingidos os seguintes objetivos:

- Realização de um estudo sistemático dos enfoques, categorias e técnicas de análise orientada a objetos.
- Estabelecimento de um conjunto de critérios objetivos e válidos para avaliar e comparar as técnicas.
- Apresentação de uma comparação das técnicas com base nos critérios definidos.
- Destaque dos pontos fracos e pontos fortes que apresentam os diversos enfoques, categorias e técnicas.

No *capítulo 1* deste trabalho são apresentados os conceitos básicos que permitem entender o que é a análise no desenvolvimento de software, o que é o paradigma de desenvolvimento orientado a objetos e, em conseqüência, o que é a análise orientada a objetos.

O *capítulo 2* apresenta inicialmente algumas taxonomias de técnicas de análise orientada a objetos, e encerra com a descrição das técnicas mais representativas das categorias descritas.

O *capítulo 3* do trabalho mostra a estrutura e descrição dos critérios que serão utilizados para realizar o estudo comparativo.

O estudo central do trabalho é apresentado no *capítulo 4*, onde são comparadas as técnicas descritas no capítulo 2 usando os critérios desenvolvidos no capítulo 3. São usadas tabelas para sintetizar e conseguir uma melhor compreensão do conteúdo da comparação.

No *capítulo 5* são indicados os pontos fracos e os pontos fortes que foram identificados nas técnicas, graças ao estudo comparativo.

Finalmente o *capítulo 6* apresenta as conclusões e alguns comentários em relação ao estudo realizado.

1. CONCEITOS BÁSICOS

Existe atualmente na literatura uma grande efervescência em torno do denominado paradigma de desenvolvimento de software orientado a objetos. Isto tem repercutido tanto nos aspectos de implementação em linguagens de programação ou em bancos de dados, como nas técnicas de modelagem, análise e projeto.

O interesse central deste trabalho está no que se conhece como *análise orientada a objetos*. Para definir exatamente o que entende-se por este tipo de análise é preciso definir inicialmente o que é a análise no desenvolvimento de software.

1.1. A ANÁLISE NO DESENVOLVIMENTO DE SOFTWARE

O desenvolvimento de software é o processo através do qual é construída uma solução, implementada no domínio computacional, a um problema em um domínio de aplicação específico do mundo real. Ou em termos mais sucintos, o desenvolvimento de software nada mais é do que um complexo mapeamento entre dois domínios.

É natural dividir este processo de desenvolvimento em fases, segundo diferentes critérios tais como a natureza específica das tarefas a desenvolver, o domínio da aplicação e o particionamento da documentação. Porém, a maioria dos autores na literatura concordam em que é possível distinguir três tarefas fundamentais no desenvolvimento de software: análise de requisitos, projeto de software e implementação computacional.

A análise de requisitos é a fase inicial que lida com o mundo real. Nesta fase é construído um modelo que representa a situação do domínio do problema considerando os requisitos próprios da solução. A análise descreve *o que* deve ser a solução para o problema e não *o como* ela será implementada.

As principais áreas com que trata a análise no desenvolvimento de software, segundo [PRE 87], são:

1. Reconhecimento do problema no domínio da aplicação.
2. Avaliação do problema e modelagem da solução.

3. Especificação da solução.
4. Revisão da especificação.

Existe uma multiplicidade de metodologias e técnicas com diversos graus de formalidade para esta fase. Uma classificação simplificada considera os seguintes enfoques:

- Metodologias orientadas às funções: o domínio da problema é modelado com maior ênfase nas funções ou processos de transformação que se comunicam por meio de fluxos ou depósitos de dados. As metodologias mais representativas deste enfoque são as pertencentes ao desenvolvimento estruturado: a análise estruturada *clássica* de DeMarco [DEM 78] e Gane & Sarson [GAN 82], a análise essencial de sistemas [MCM 91], e em menor grau, a análise estruturada *moderna* de Yourdon [YOU 90].

- Metodologias orientadas a dados: o domínio da problema é modelado privilegiando as estruturas de dados e os relacionamentos entre elas. Os processos são modelados como operadores sobre estas estruturas. As metodologias mais representativas deste enfoque são a modelagem de informação de Flavin [FLA 81], o desenvolvimento de sistemas de Jackson *JSD* (*Jackson Systems Development*) [JAC 83], e a engenharia da informação de Martin [MAR 90].

O projeto de software, por sua parte, trata com a especificação do domínio do problema construída na análise e a tecnologia computacional disponível. Nesta fase o modelo da análise é traduzido em uma representação no domínio computacional. Nesta fase a ênfase é colocada em *como* será implementada a solução. Os enfoques indicados acima também são aplicáveis nesta fase.

Finalmente, a implementação trata da representação computacional do projeto e as linguagens e técnicas de programação. Nesta fase é feita a codificação, teste e integração dos componentes de software que configuram a solução computacional do problema.

Ao considerar o paradigma de orientação a objetos, uma boa parte dos conceitos do desenvolvimento *tradicional*[1] -como alternativa ao orientado a objetos- são mantidos, outros são modificados e muitos são acrescidos, mudando a óptica com que o desenvolvimento é conduzido.

1.2. A ORIENTAÇÃO A OBJETOS

O termo orientação a objetos, que tem surgido como um novo paradigma no desenvolvimento de software, significa que o software é estruturado e construído como um conjunto de entidades denominadas objetos.

Os principais conceitos do paradigma de orientação a objeto são:

- *Identidade* significa que os dados são discretizados em entidades identificáveis chamadas de *objetos*.
- *Encapsulamento* é a capacidade que possuem os objetos de incorporar tanto as estruturas de dados que os determinam, como as operações aplicáveis a estas estruturas. Estas estruturas são definidas como os atributos do objeto, e as operações como os métodos do objeto. Os atributos e os métodos são propriedades exclusivas e intransferíveis dos objetos.
- *Classificação* é o processo que permite que objetos que possuem uma mesma estrutura de dados (atributos) e um mesmo conjunto de operações (métodos) sejam agrupados em *classes*. Uma classe então é uma abstração que representa as principais propriedades comuns de um conjunto de objetos em um domínio de aplicação, e ignora as diferenças irrelevantes para este domínio.
- *Polimorfismo* implica que uma mesma operação pode comportar-se de maneira diferente em classes distintas, a pesar de possuir o mesmo nome. Por exemplo, *inverter* uma matriz é uma operação que trabalha de maneira muito distinta à operação de *inverter* uma figura geométrica.

[1] Em geral, quando utilizado o termo *desenvolvimento tradicional*, se está fazendo referência àquele desenvolvimento que utiliza a decomposição funcional ou *top-down* como estratégia de particionamento.

- *Herança* é entendida como o compartilhamento de atributos e/ou métodos entre classes relacionadas por hierarquias de generalização e/ou especialização. Diferentes classes podem ser agrupadas em uma *superclasse* mais genérica que descreve as propriedades comuns das classes, ou uma classe genérica pode ser especializada e refinada em *subclasses*. As propriedades herdadas da superclasse às classes, e da classe às subclasses não precisam ser repetidas em cada classe. Existe também um tipo de herança particular denominada *herança múltipla*, que permite que uma classe possa herdar de mais de uma superclasse ao mesmo tempo.

Uma boa introdução aos conceitos fundamentais deste paradigma pode ser encontrada em [BOO 91], [GIR 90], [KOR 90], [RIN 91], [RUM 91] e [WIR 90].

Neste trabalho será usado o termo *classe* para indicar o conjunto de entidades com atributos e/ou métodos comuns, uma *instância* significa uma realização de uma classe (uma das entidades da classe), e um *objeto* indica indistintamente uma classe ou uma instância da classe (como na linguagem *Smalltalk*).

1.3. O DESENVOLVIMENTO ORIENTADO A OBJETOS

A metodologia de desenvolvimento de software orientado a objetos tem como essência a identificação e organização dos conceitos do domínio da aplicação, e consiste na construção de um modelo de objetos deste domínio, o qual é mapeado no domínio computacional acrescentando detalhes de implementação nas sucessivas fases do desenvolvimento [RUM 91].

As grandes fases próprias do desenvolvimento tradicional de software também podem ser identificadas no contexto do paradigma de orientação a objetos, mas com algumas redefinições dos papéis e das relações entre estas fases. É possível definir, em conseqüência, a análise orientada a objetos, o projeto orientado a objetos e a implementação orientada a objetos.

Na análise orientada a objetos é construído um modelo que representa a situação do domínio do problema em termos de objetos e suas propriedades. Estes obje-

tos são conceitos próprios do domínio do problema e não implementações de tais conceitos [RUM 91].

O projeto orientado a objetos consiste na construção de um modelo do projeto baseado no modelo da análise, incluindo detalhes de implementação e objetos próprios do domínio da solução (o domínio computacional). O fundamental nesta fase são as estruturas de dados e os algoritmos necessários para implementar as estruturas [RUM 91].

A implementação orientada a objetos consiste na tradução dos objetos e relacionamentos estabelecidos na análise e no projeto em uma linguagem de programação específica, em um banco de dados ou em um hardware particular [RUM 91].

No modelo final a ser implementado, são representados os objetos do domínio da aplicação e do domínio computacional com os mesmas notações, ainda que estes objetos pertençam a níveis de abstração diferentes.

Se comparado o desenvolvimento orientado a objetos em relação ao desenvolvimento tradicional, pode indicar-se como diferença básica entre eles a estratégia de particionamento, isto é, a forma em que se avança no processo de desenvolvimento desde os níveis mais abstratos aos mais detalhados (passo do *macro* ao *micro*): o desenvolvimento tradicional baseia-se na decomposição funcional e o desenvolvimento orientado a objetos baseia-se na decomposição por objetos. No primeiro caso, o particionamento é feito da função para as subfunções, onde as subfunções, ao desempenharem seu papel, permitem a completação da função; no segundo caso, o particionamento é realizado de acordo com hierarquias de generalização e especialização de classes, com herança de atributos e métodos[2].

A mesma estratégia pode ser vista através de uma óptica distinta: o princípio da agregação das funções, que pode ser entendido como um processo inverso ao do particionamento, isto é do *micro* ao *macro*. No desenvolvimento tradicional, funções são agrupadas se elas constituem passos na execução de uma função de mais alto nível. Na orientação a objetos as funções são agrupadas juntas se elas operam sobre uma mesma abstração de dados: o objeto [BAI 89].

---

[2] Também podem ser consideradas, em menor grau por não apresentarem herança, as hierarquias de agregação.

Outra diferença relevante é que os modelos, e em conseqüência as primitivas destes modelos, são os mesmos nas diferentes fases, i.e. não é preciso realizar nenhum tipo de tradução ou conversão na medida em que se avança no processo de desenvolvimento. Ou em outras palavras, se é definido o *gap semântico* como a distância entre o domínio do problema no mundo real e o domínio da solução na implementação computacional, este *gap* se vê diminuído ao utilizar o paradigma da orientação a objetos. Assim por exemplo, os objetos identificados no domínio do problema são mapeados diretamente nos objetos especificados no domínio da solução, e o mesmo acontece com estes objetos ao serem mapeados nos objetos codificados na implementação. Isto é possível graças à homogeneidade ou uniformidade conceitual nas sucessivas fases do desenvolvimento orientado a objetos. A figura 1.1 mostra estes mapeamentos.

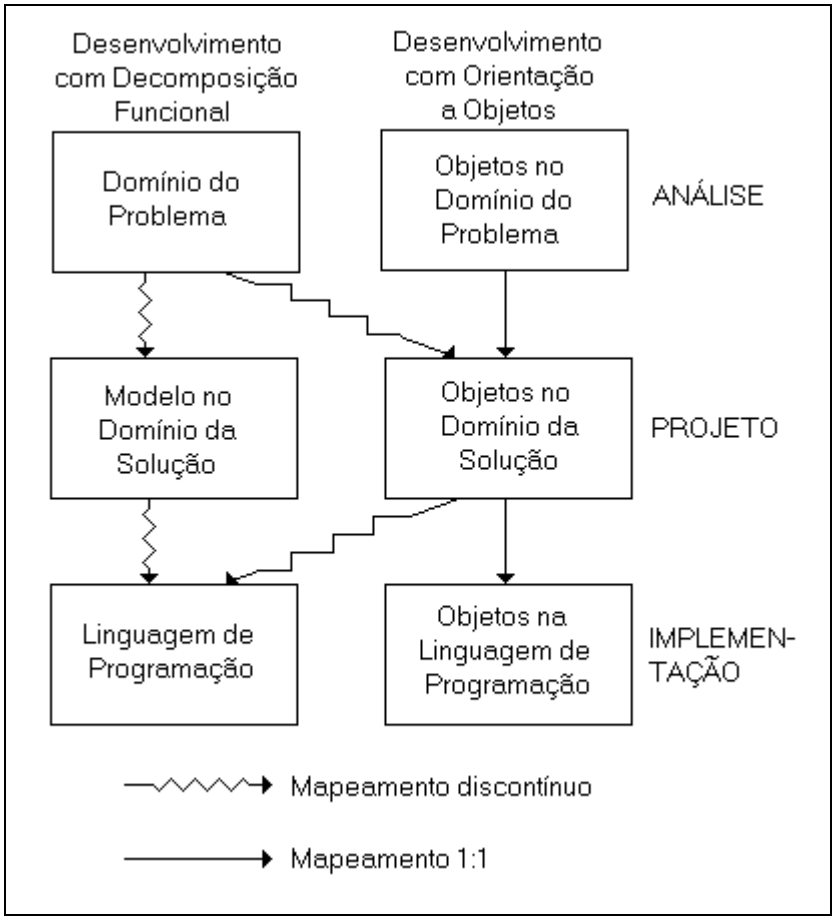

Figura 1.1 - Mapeamentos comparativos entre as fases dos desenvolvimentos tradicional e orientado a objetos (adaptado de [HEN 90a]).

Em termos práticos, o desenvolvimento orientado a objetos promete um alto grau de reusabilidade e, como conseqüência, uma maior produtividade. Segundo Yourdon [YOU 90a]: " ... no desenvolvimento tradicional tende-se a tratar cada novo projeto como um exercício intelectual nunca antes contemplado; análise e projeto começam literalmente com uma folha de papel em branco. Por outra parte, ao usar métodos orientados a objetos, a tendência é usar *projeto por extensão*. Isto é, assume-se que o novo sistema a desenvolver corresponde simplesmente a uma extensão ou refinamento de algo que já tinha sido construído ... ".

Outras vantagens são uma maior extensibilidade, flexibilidade, manutenibilidade e robustez em relação ao desenvolvimento tradicional. Estes e outros fatores de qualidade são detalhados em [MEY 88] e [KOR 90].

Como este trabalho tem na análise orientada a objetos o tema principal de tratamento, é preciso detalhar esta fase e sua relação com as outras fases no desenvolvimento.

1.4. A ANÁLISE ORIENTADA A OBJETOS (AOO)

A AOO não pode ser classificada inteiramente em nenhum dos enfoques metodológicos indicados anteriormente (seção 1.1): não é orientada às funções nem a dados. Nestes termos, a análise concebida sob o paradigma de orientação a objetos representa uma mudança radical com respeito ao enfoque funcional, e uma mudança evolucionária ou incremental com respeito ao enfoque de dados [FIC 92]. A AOO não é orientada à função porque a decomposição funcional é ortogonal aos conceitos de encapsulamento (as operações são *serviços* exclusivos dos objetos), classificação e herança. A AOO não é puramente orientada a dados devido também ao princípio de encapsulamento. No entanto ela baseia-se em grande parte na modelagem semântica de dados ou modelagem de informação, onde os conceitos de classificação e herança já aparecem.

Outro aspecto que deve ser destacado é a dificuldade para distinguir precisamente entre a AOO e o projeto orientado a objetos (POO) na literatura. Esta dificuldade é mais obvia quando se busca determinar exatamente onde termina a AOO e onde começa o POO, já que a divisão de fronteiras entre estas fases é confusa [MON 92]. Esta falta de clareza é o resultado da uniformidade conceitual ou continuidade entre as fases do desenvolvimento [KOR 90]. Este fenômeno é refletido claramente pelo modelo chafariz (*fountain*) do ciclo de vida da orientação a objetos mostrado por Henderson-Sellers [HEN 90a] (ver figura 1.2), em contraposição ao modelo cascata (*waterfall*) tradicional. O processo de desenvolvimento orientado a objetos consiste em uma estratégia iterativa em que os objetos são identificados e definidos, conjuntamente com seus relacionamentos e comunicações. No avanço do processo de desenvolvimento, podem ser encontradas novas classes ou serem redefinidas algumas das existentes, isto resulta em que novos relacionamentos, comportamentos e comunicações devem ser definidos [MON 92]. Isto implica uma grande sobreposição e iteração entre as fases do desenvolvimento, onde a analise e o projeto não são uma exceção.

Um critério útil para diferenciar a AOO do POO é o domínio de operação: a AOO trabalha com o domínio do problema ou *universo do discurso*, enquanto que o POO trabalha com o domínio da solução ou projeto de implementação computacional.

Para os efeitos deste trabalho, a AOO será definida como *a construção de modelos do domínio do problema, identificando e especificando um conjunto de objetos semânticos que interagem e comportam-se conforme os requisitos do sistema* [MON 92]. Os objetos semânticos são aqueles que possuem um significado específico no domínio do problema.

É interessante destacar que, de acordo com a definição anterior, e considerando a importância do conceito da reusabilidade na orientação a objetos, a AOO parece estar muito relacionada com o que se conhece como *Análise de Domínio* [ARA 89], [GOS 90] e [ISC 91], segundo Neighbors (citado por [ARA 89] p.153), corresponde a “ ... uma tentativa de identificar os objetos, operações e relacionamentos entre o que os especialistas do domínio [do problema] percebem ser importante sobre este domínio ”. De Champeaux [DEC 92] acrescenta a necessidade de incluir na análise de domínio, o estudo de sistemas *semelhantes* ou relacionados, com o propósito de encontrar

potenciais conceitos, componentes ou arquiteturas para o reuso. A análise de domínio então, parece preceder com algum grau de sobreposição o que foi definido como AOO.

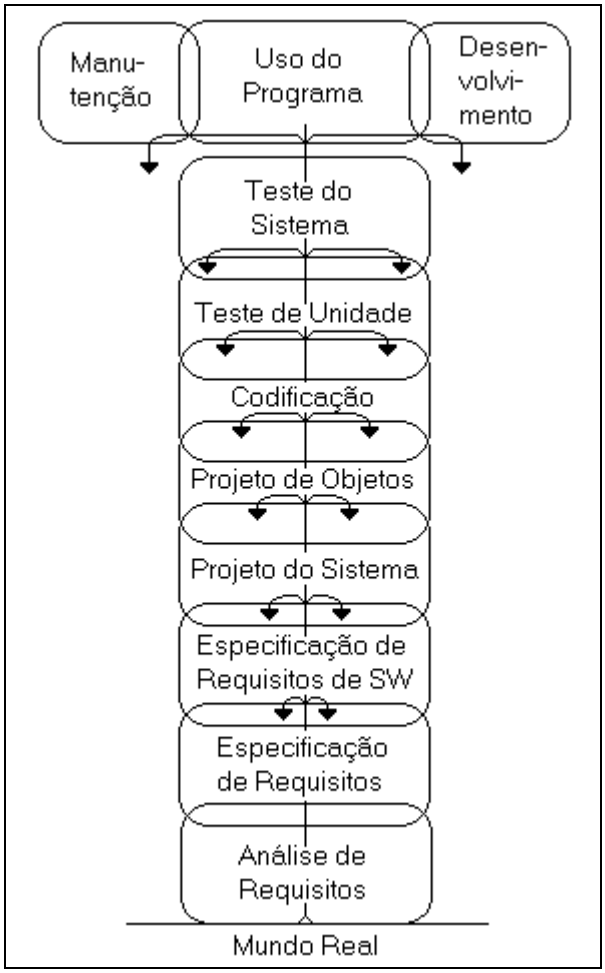

Figura 1.2 - Modelo chafariz (*fountain*) para o ciclo de vida do software orientado a objetos [HEN 90a].

Para complementar a definição dada acima sobre a AOO, o POO será definido como *a modelagem do domínio da solução, que inclui as classes semânticas especificadas na análise (com possíveis acréscimos) e classes de interface, de aplicação e genéricas*[3] *identificadas durante o processo de projeto* [MON 92]. As classes de inter-

[3] No artigo de Monarchi & Puhr [MON 92], os autores utilizam o nome de *básicas/utilitárias* (*base/utility*) para referir-se a este tipo de classes, porém parece mais apropriado o termo *classes*

face são aquelas relacionadas com a interface da aplicação e não são parte do domínio do problema. As classes de aplicação são *drivers* ou mecanismos de controle do sistema, como por exemplo os menus. Uma classe genérica é aquela que é independente dos domínios do problema e da solução, como por exemplo uma pilha, um *string* ou um *debugger*.

Conforme o exposto até aqui, é fácil deduzir que a AOO é essencialmente baseada na modelagem. É de se esperar então, que a especificação resultante da aplicação de técnicas de AOO resulte em múltiplos modelos e múltiplas notações [DEC 92]. Nesta perspectiva, o processo de construção dos modelos do domínio do problema, que é a razão de ser da AOO, deve considerar diferentes aspectos ou pontos de vista. Estes aspectos constituem-se nas dimensões da modelagem orientada a objetos.

1.5. AS DIMENSÕES DA MODELAGEM ORIENTADA A OBJETOS

Toda modelagem orientada a objetos exige, no mínimo, dois aspectos ortogonais ou dimensões para descrever um sistema relativamente complexo: a dimensão estrutural dos objetos e a dimensão dinâmica do comportamento. Pode ser considerada também uma dimensão adicional: a dimensão funcional dos requisitos, que tem gerado uma grande controvérsia na literatura.

Tanto a modelagem dinâmica quanto a modelagem funcional descansam sobre a modelagem estrutural como mostra a figura 1.3. Contudo, existe uma discussão em torno da forma em que deve ser realizada a modelagem na dimensão funcional e sua relação com os outros aspectos da modelagem orientada a objetos.

Nas próximas seções serão definidas e discutidas estas dimensões, incluindo a dimensão funcional.

---

genéricas considerando o fato de que trata-se de tipos como *strings* ou *arrays*, que nada mais são do que tipos genéricos de ampla aplicabilidade.

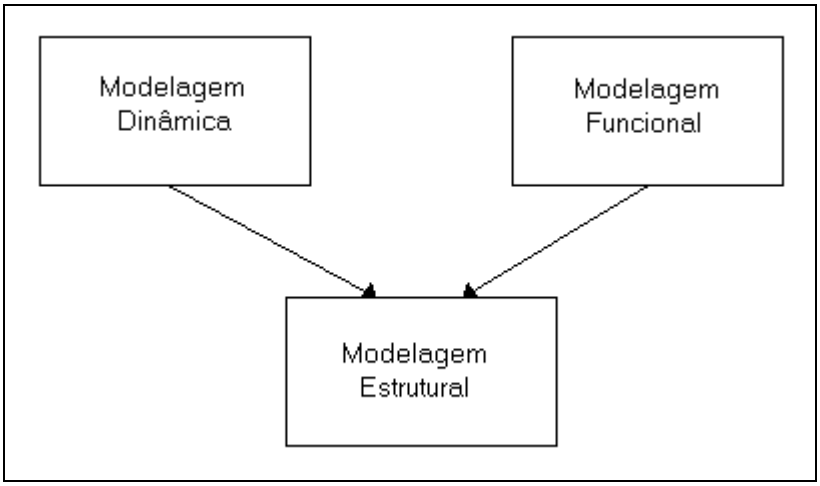

Figura 1.3 - A modelagem estrutural como base na orientação a objetos.

1.5.1. A Dimensão Estrutural dos Objetos

Este dimensão centra-se no aspecto estático ou passivo, por isto está relacionado com a estrutura estática dos objetos que fazem parte do sistema. A estrutura reflete a *identidade* de cada objeto, sua *classificação*, seu *encapsulamento* (seus *atributos* e seus *métodos*) e seus *relacionamentos* estáticos (hierarquias de generalização e especialização, agregação, composição e associações específicas como *cria*, *usa*, etc.).

Por exemplo, o seguinte enunciado descreve alguns elementos que fazem parte da dimensão estrutural dos objetos para um sistema de recursos humanos: a classe *Empregado*, que é uma subclasse da classe *Pessoa* herdando os atributos de *Nome* e *Endereço*, possui os atributos próprios de *Código Empregado*, *Estado Empregado* e *Salário*, e os métodos próprios *Contrata Empregado*, *Libera Empregado* e *Modifica Salário*, e está associado a uma classe *Departamento*, através de uma associação específica denominada *Empregado Trabalha em Departamento*.

Esta dimensão é fundamental na modelagem orientada a objetos porque constitui a base sobre a qual o(s) outro(s) aspecto(s) trabalha(m). E é por isto que geralmente é recomendado começar a modelagem orientada a objetos considerando esta dimensão antes da(s) outra(s).

Em relação à dimensão dinâmica, a dimensão estática especifica as classes e atributos que irão mudar (sofrendo transições de estados de acordo com eventos). Em se tratando da dimensão funcional, esta dimensão modela os métodos que transformarão, e as classes e atributos que sofrerão tais transformações.

1.5.2. A Dimensão Dinâmica do Comportamento

A dimensão dinâmica tem a ver com o aspecto dinâmico ou ativo, por isto descreve o comportamento dos objetos que constituem os sistemas. O comportamento é refletido por meio de *estados* (representados por valores dentro dos domínios específicos dos atributos), *transições* entre estes estados (mudança dos valores), *eventos* (fatos que ocorrem e que produzem as transições) e *ações* (representados pelos métodos dos objetos, podendo ocorrer durante as transições ou permanência nos estados).

Ao modelar considerando este aspecto é importante identificar os eventos que produzem as mudanças, as seqüências dos eventos, os estados que definem o contexto para estes eventos, e a organização destes eventos, estados e ações. Assim visto, a dimensão dinâmica descreve o controle, isto é, as seqüências de eventos que determinam o comportamento do sistema.

Utilizando o mesmo exemplo do sistema de recursos humanos, na ocorrência do evento *Início de Férias*, o estado (atributo) *Estado Empregado* pode mudar do valor *Ativo* para o valor *Em Férias*. Ou se ocorrer o evento *Incremento de Salário* é disparada a ação (método) *Modifica Salário*.

É importante destacar que esta dimensão utiliza os atributos, métodos e classes previamente identificados na modelagem estrutural para configurar o comportamento dos objetos do sistema.

1.5.3. A Dimensão Funcional dos Requisitos

Nesta dimensão da modelagem é considerado o aspecto relativo à função de transformação global do sistema, i.e. à conversão de entradas em saídas. Esta transformação global é refletida através de processos[4] ou *funções* (que transformam valores) e *fluxos de dados* (entradas e saídas destas funções), configurando com isto redes funcionais. Geralmente esta modelagem é particionada utilizando a estratégia *top-down* ou decomposição funcional por refinamento sucessivo.

Continuando com o exemplo do sistema de recursos humanos, um requisito para este sistema poderia ser que a ocorrência de um incremento salarial implicar uma entrada de valores percentuais e cotas mínimas e máximas para a aplicação de tais percentuais, uma função de cálculo dos novos valores para *Salário* (atributo) e a aplicação do método *Modifica Salário* para atualiza-lo para cada *Empregado* (objeto), e uma saída de um relatório com o *Nome* e novo *Salário* (atributos) de cada *Empregado* (objeto).

A ferramenta mais comum utilizada para esta modelagem é o *diagrama de fluxo de dados* ou *DFD*, que foi divulgada pelas técnicas estruturadas de análise, e é descrito na sua versão *moderna* em [YOU 90]. Nestes diagramas, além dos processos e fluxos de dados são especificados os depósitos de dados (fluxos de dados detidos temporariamente) e entidades externas ao sistema que geram ou recebem fluxos de dados. O particionamento dos DFD's é realizado através da decomposição *top-down*.

Na modelagem funcional *clássica*, as funções são identificadas no contexto do sistema, e não em relação aos objetos constituintes do sistema. O conceito de transformação ou processo global não dependente dos objetos individuais parece contrariar a filosofia da orientação a objetos. Uma modelagem deste tipo violaria o princípio básico de encapsulamento dos objetos, porque as funções (métodos) poderiam acessar diretamente múltiplos objetos sem a subordinação própria do encapsulamento [FIC 92].

Segundo Booch e De Champeaux (citados por [FIC 92] p.38), os modelos de fluxos de dados não são recomendados porque inevitavelmente irão produzir uma *orientação funcional* no sistema de objetos.

---

[4] Nesta e outras seções, serão usados indistintamente os termos de *função*, *processo* e *transformação*.

Contudo, não pode ser desconhecido o fato, de que em muitos domínios de problema são necessárias transformações globais que afetam muitos objetos, e comprometem a execução seqüencial ou paralela de numerosos passos intermediários entre seu início e término [FIC 92]. Ou em outras palavras, parece difícil conceber requisitos para sistemas de software em termos não funcionais, já que é comum referir-se a eles justamente como requisitos funcionais (saídas desejadas a partir de entradas disponíveis).

De acordo com Jalote [JAL 89]: " ... os enfoques de projeto orientado a objetos que dão pouca ênfase na função de transformação global, assumem que a função está definida uma vez que os objetos e seus métodos são especificados. Esta suposição pode ser válida para sistemas pequenos, mas parece não ser verdadeira para problemas complexos, onde a função pode ser bastante complexa. ". O autor propõe então que uma técnica orientada a objetos em conjunto com um enfoque funcional constituem um enfoque mais geral, que permitirá abordar efetivamente a complexidade de grandes problemas.

Esta contradição que parece existir em torno da modelagem funcional pode ser resolvida considerando as seguintes situações:

- Se o modelo funcional for construído *antes* ou *independente* da modelagem estrutural dos objetos, a situação corresponde à modelagem funcional clássica de sistemas, onde não é considerado o encapsulamento dos métodos nos objetos. Neste sentido, um sistema pode ser modelado como um conjunto de objetos que interagem entre eles, e uma função de transformação que opera sobre os objetos para realizar a computação requerida [JAL 89].

- Se a modelagem funcional for realizada *após* o aspecto estático ser modelado, isto é, se o modelo funcional for construído considerando ou dependendo da modelagem estrutural dos objetos, então pode ser respeitado princípio de encapsulamento dos métodos. As funções novas podem ser *alocadas* como métodos nos objetos previamente identificados e especificados, e as funções restantes podem ser escolhidas do conjunto de métodos disponíveis.

A discussão em torno das duas situações expostas acima ainda não está esgotada na literatura, mas parece que a tendência favorece à última proposta. Em [BAI 89] e [FIC 92] é sugerida a modelagem de processos *end-to-end* que represente a rede de serviços encapsulados (métodos) mostrando seqüência, paralelismo e execução condicionada, isto corresponde exatamente a uma modelagem funcional subordinada aos objetos.

A descrição separada que foi feita das dimensões não significa que necessariamente sejam construídos modelos separados para cada aspecto. Pode ser modelado mais de um aspecto simultaneamente, sem prejuízo das considerações aqui mencionadas.

Finalmente, como conclusão é possível afirmar que as três dimensões descritas mostram aspectos que são importantes para a modelagem orientada a objetos, e por tanto deveriam ser consideradas pelas técnicas que observam este paradigma. A figura 1.4 sintetiza esta noção.

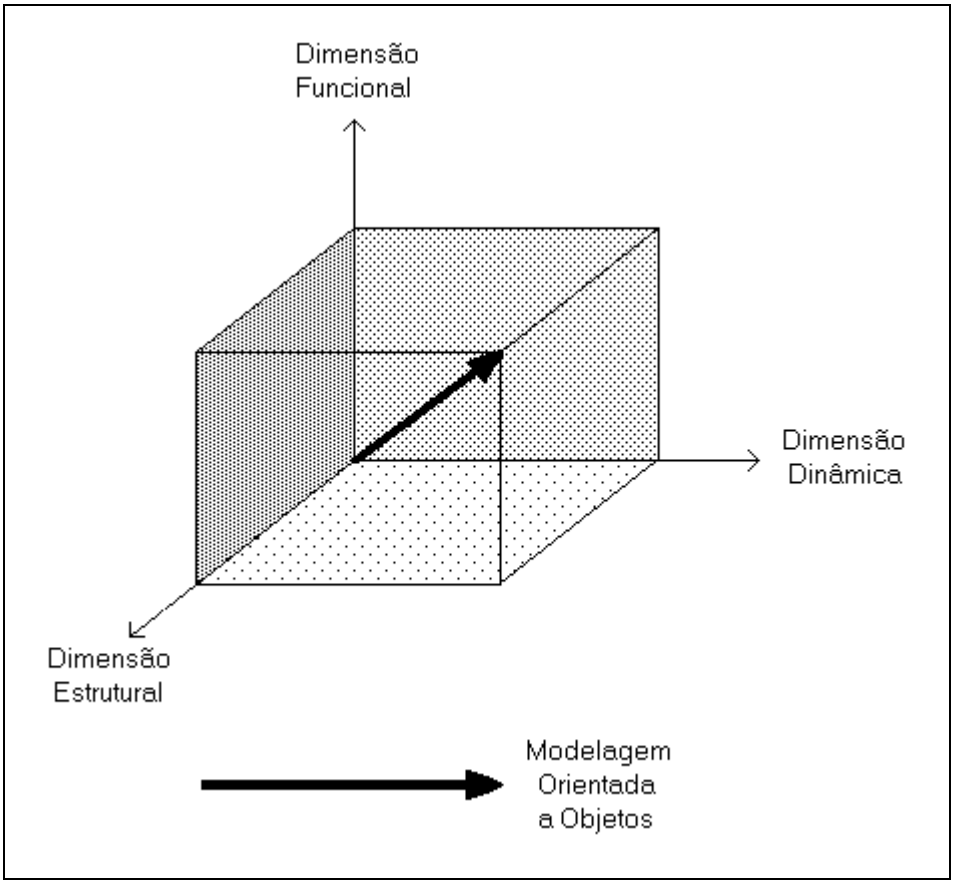

Figura 1.4 - A modelagem orientada a objetos como enfoque multidimensional (adaptado de [PRI 91]).

Uma vez definidos os conceitos básicos para compreender a natureza da AOO, o capítulo seguinte apresenta uma revisão das técnicas mais significativas dentro deste paradigma de orientação a objetos.

2. REVISÃO DAS TÉCNICAS DE ANÁLISE ORIENTADA A OBJETOS

Este capítulo tem como propósito apresentar algumas taxonomias que serão úteis para posteriormente poder descrever e comparar as técnicas de AOO.

2.1. AS CLASSIFICAÇÕES DE TÉCNICAS DE AOO

As classificações que serão revistas inicialmente são: a *classificação das metodologias existentes orientadas a objetos* de Capretz & Lee [CAP 92], e os *enfoques gerais para a AOO e o POO* de Monarchi & Puhr [MON 92]. Depois é proposta uma classificação com um critério de maior granularidade.

2.1.1. A Classificação das Metodologias Existentes Orientadas a Objetos

Em [CAP 92] é proposta uma classificação que distingue duas direções nas metodologias de orientação a objetos:

- *Adaptação*: Esta direção está relacionada com a mescla de um enfoque orientado a objetos com as metodologias bem conhecidas de desenvolvimento estruturado de sistemas.

- *Assimilação*: Esta direção dá ênfase no uso de um enfoque orientado a objetos para o desenvolvimento de sistemas de software, porém observando o tradicional modelo cascata de ciclo de vida do software.

A *adaptação* pode ser entendida como a inclusão de modelos funcionais do tipo dos DFD's como uma especificação inicial em uma metodologia orientada a objetos. Esta combinação busca a complementaridade das técnicas estruturadas com a tecnologia emergente da orientação a objetos, pagando o preço da subutilização da potencialidade do novo paradigma.

Esta estratégia pode ser válida como um enfoque de transição entre ambas as filosofias: a estruturada e a da orientação a objetos [CAP 92] e [HEN 90].

Já no caso da *assimilação*, a tendência é mudar somente as técnicas de análise, projeto e implementação mas preservando a linhas básicas do ciclo de vida tradicional. Por isto é dito que o ciclo de vida tradicional *assimila* o novo paradigma.

Contudo, a maioria das propostas na literatura ainda não conseguiu cobrir com sucesso todo o ciclo de vida. Falta um maior amadurecimento e experiência no desenvolvimento de grandes projetos para poder avaliar e melhorar tais propostas.

Finalmente, ainda que esta classificação tenha sido concebida para metodologias de orientação a objetos, é possível aplica-la, pelo menos parcialmente, às técnicas de AOO.

2.1.2. Os Enfoques Gerais para a AOO e o POO

Em [MON 92] é descrita uma classificação que utiliza como critério básico a forma e o grau em que as técnicas orientadas a objetos incorporam outros paradigmas. Os enfoques são:

- *Enfoque Combinativo*: É aquele que utiliza as técnicas orientada a objetos, orientada à função e/ou orientada à dinâmica para modelar separadamente a estrutura, a funcionalidade e/ou o comportamento dinâmico e fornece métodos para integrar estes diferentes modelos. Em outras palavras, corresponde às técnicas que tratam separadamente as dimensões da modelagem orientada a objetos. Este enfoque pode ver-se seriamente deteriorado na integração necessária para a transição à fase de projeto.

- *Enfoque Adaptativo*: É aquele que usa as técnicas existentes em uma nova (orientada a objetos) maneira, ou estende as técnicas existentes para incluir a orientação a objetos. Neste enfoque é possível *adaptar* procedimentos e notações que já provaram ser úteis, na perspectiva do novo paradigma. Por exemplo, os tradicionais DFD's ou outras representações de orientação funcional podem ser usados no novo contexto, representando a funcionalidade dos objetos e não a funcionalidade do sistema. Tácticas

semelhantes podem ser aplicadas aos modelos semânticos de dados, modelos entidade-relacionamento estendidos, ou diagramas de transição de estados.

- *Enfoque Puro*: É aquele que usa técnicas novas para modelar a multidimensionalidade da orientação a objetos. Neste enfoque os objetos são modelados no domínio do problema, conjuntamente com seus relacionamentos no sistema, sem requerer traduções de nenhuma espécie. Mas precisam de sofisticados mecanismos de gerenciamento da complexidade para permitir visões em vários níveis de abstração e detalhe ou desde o ponto de vista dos aspectos da modelagem.

2.1.3. Uma Classificação Específica para as Técnicas de AOO

A classificação proposta utiliza como critério básico a essência da origem da técnica, isto é, o critério responde à pergunta: a partir de quais conceitos originou-se a técnica? Isto pode ser verificado observando a ênfase que determinadas técnicas de AOO outorgam a conceitos, aspectos, procedimentos ou representações em uma ou mais dimensões de modelagem.

Conforme o critério acima, as técnicas podem ser classificadas em: textuais, evolutivas, integracionistas, reversas e comportamentais. A figura 2.1 mostra a estrutura desta classificação.

1. *Técnicas Textuais*: são aquelas que baseiam-se em descrições informais, porém precisas, escritas em linguagem natural para, através de uma análise sintática de nomes, adjetivos, verbos e advérbios, identificar objetos, atributos, operações e relacionamentos tanto do domínio do problema como do domínio da solução. Esta categoria de técnicas teve sua origem fora do paradigma de orientação a objetos, especificamente no trabalho de Abbott [ABB 83] que propõe projetar programas em *Ada* a partir de descrições informais em inglês. Hoje estas técnicas são insuficientes para abordar problemas mais complexos e podem ser consideradas como ultrapassadas, porém são indicadas aqui pela sua relevância histórica.

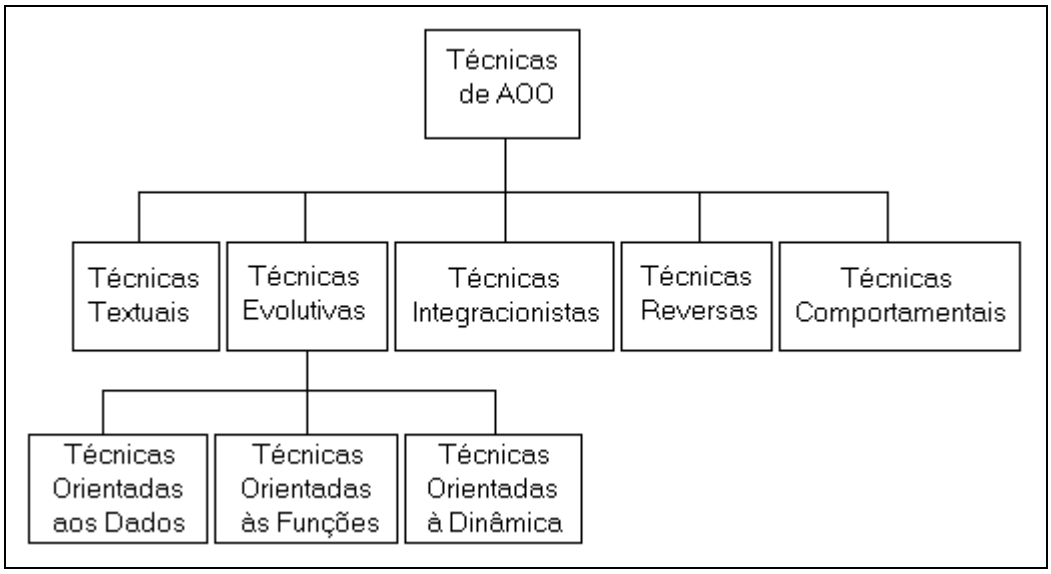

Figura 2.1 - A estrutura proposta de classificação de técnicas de AOO.

2. *Técnicas Evolutivas*: são aquelas produto da extensão ou evolução de técnicas com orientação para algumas das dimensões da modelagem e a complementação com outros aspectos de modelagem. Esta categoria de técnicas pode ainda ser subdividida, segundo a orientação que apresentem, em:

    - *Técnicas Orientadas a Dados*: são aquelas concebidas como evoluções semânticas de modelos de dados, geralmente em torno do modelo de dados entidade-relacionamento [CHE 76] e suas extensões, por exemplo a apresentada em [TEO 86].

    - *Técnicas Orientadas às Funções*: são aquelas generalizadas a partir de modelos funcionais, i.e. aquelas que utilizam como base modelos de tipo funcional com complementos ou transformações no paradigma de orientação a objetos. Por ser a ferramenta mais conhecida, os DFD's são os mais usados [YOU 90], por citar a sua versão mais *moderna*.

    - *Técnicas Orientadas à Dinâmica*: são aquelas que estendem ou complementam modelos dinâmicos, como por exemplo redes de Petri [HEU 90]

ou *statecharts* [HAR 87]. O complemento indispensável é algum modelo estático.

3. *Técnicas Integracionistas*: são aquelas que combinam de maneira balanceada modelos nas diversas dimensões (estrutural, dinâmica e funcional), para integra-los na especificação orientada a objetos.

4. *Técnicas Reversas*: são aquelas originadas a partir de necessidades de implementação, i.e. como suporte para os conceitos de linguagens específicas de orientação a objetos, como por exemplo: *Simula, Smalltalk, Ada*[5] ou *Eiffel*. Estas técnicas seguem a mesma evolução do paradigma de orientação a objetos: tiveram sua origem em linguagens de programação orientadas a objetos, que demandaram técnicas de projeto orientado a objetos, que, por sua vez, exigiram o mesmo paradigma a nível da análise.

5. *Técnicas Comportamentais*: são aquelas originadas na identificação e modelagem concebidas a partir do comportamento global esperado do sistema de objetos. São técnicas que derivam os objetos internos do sistema a partir do comportamento externo do sistema. Complementam posteriormente os modelos com outros aspectos.

2.1.4. A Comparação das Classificações

Interessa principalmente as relações que podem ser estabelecidas entre a classificação proposta e as outras duas referenciadas. A tabela 2.1 resume estas relações.

Segundo o critério da classificação proposto serão descritas as técnicas mais representativas de cada categoria na seguinte seção.

2.2. AS TÉCNICAS DE AOO MAIS REPRESENTATIVAS

[5] Este é um caso singular, já que *Ada*, que não é uma linguagem de orientação a objetos, pois carece de suporte para o conceito fundamental de herança, é a linguagem para a qual tem sido desenvolvidas mais técnicas de análise e projeto orientados a objetos.

Entender-se-á como técnica de AOO àquela técnica que ajusta-se à definição dada na seção 1.4 deste trabalho. É fundamental então que o processo de modelagem proposto seja descrito com um grau razoável de detalhe, para permitir sua comparação com outras técnicas. O fato de se a técnica possui ou não representações dos modelos é considerado apenas na comparação.

	METODOLOGIAS EXISTENTES ORIENTADAS A OBJETOS (CAPRETZ & LEE)		ENFOQUES GERAIS PARA ANÁLISE E PROJETO ORIENTADOS A OBJETOS (MONARCHI & PUHR)		
CLASSIFICAÇÃO PROPOSTA DAS TÉCNICAS	ADAPTAÇÃO	ASSIMILAÇÃO	COMBINATIVO	ADAPTATIVO	PUROS
TEXTUAIS					
EVOLUTIVAS:	Parcialmente relacionadas			Mesma categoria	
• Orientadas a dados				Parcialmente relacionadas	
• Orientadas às funções	Mesma categoria			Parcialmente relacionadas	
• Orientadas àdinâmica				Parcialmente relacionadas	
INTEGRACIONISTAS			Mesma categoria		
REVERSAS					Parcialmente relacionadas
COMPORTAMENTAIS					Parcialmente relacionadas

Parcialmente relacionadas = os critérios são tais que existe uma interseção parcial entre os conjuntos de técnicas pertencentes a ambas as categorias

Mesma categoria = a interseção é total entre os conjuntos de técnicas pertencentes a ambas as categorias

Tabela 2.1 - Relações entre as classificações referenciadas e a proposta.

De acordo com a consideração acima, não foram analisadas as técnicas de análise e/ou projeto que centravam-se principalmente em representações e notações para sistemas orientados a objetos. Entre estas técnicas podem ser mencionadas: a notação gráfica de [ACK 91], os cartões de colaboração de classe-responsabilidade (*Class-Responsability Collaboration Cards* ou *CRC Cards*) de [BEC 89], os diagramas de mensagens de [CUN 86], a notação uniforme de objetos (*Uniform Object Notation* ou *UON*) de [PAG 90], a notação de projeto estruturado orientado a objetos (*Object-Oriented Structured Design Notation* ou *OOSD*) de [WAS 90], e os diagramas de classes de [WIL 90].

As técnicas que são descritas a seguir foram escolhidas por serem as mais representativas da categoria à qual pertencem. Outras técnicas da mesma categoria são referenciadas e comentadas sucintamente.

2.2.1. As Técnicas Textuais

As denominadas técnicas textuais baseiam-se em uma análise sintática de uma descrição informal escrita em linguagem natural para identificar objetos, atributos e métodos.

Como a técnica representativa pode ser indicada a *Análise Orientada a Objetos* de [PRE 87] que sintetiza outras propostas. A técnica pode ser descrita da seguinte forma:

1. O sistema é descrito usando uma estratégia informal. Esta estratégia corresponde a uma descrição em linguagem natural da solução do problema a resolver. A estratégia informal pode ser estabelecida na forma de parágrafos simples, gramaticalmente corretos.

2. Os objetos são identificados sublinhando cada *nome* ou *cláusula nominal*. Cada objeto é acrescentado a uma tabela simples. Os *sinônimos* também devem ser destacados.

3. Os atributos dos objetos são identificados sublinhando todos os *adjetivos* que são associados aos objetos respectivos (nomes).

4. Os métodos são identificados sublinhando todos os *verbos*, *frases verbais* e *predicados*. Cada método é relacionado com o objeto apropriado.

5. Os atributos dos métodos são identificados sublinhando todos os *advérbios*. Estes atributos são associados aos métodos respectivos (verbos).

A tabela 2.2 mostra um exemplo de uma tabela gerada para um sistema de classificação de caixas em uma cinta transportadora.

Objeto	**Atributo**	**Método correspondente**
Dados	Código da caixa	Decodificar
Posição	Caixa	Determinar
Sinal	Controle	Sincronizar, produzir
Entrada	Tacômetro de pulso	Receber
Lista	FIFO	Anotar
Banco de dados	1.000 entradas	Consultar
Pulsos	Contagiros	Gerado

Tabela 2.2 - Exemplo de tabela de objetos nas técnicas textuais [PRE 87].

Pode observar-se que a técnica é muito intuitiva e rudimentar, desconhecendo vários conceitos do paradigma. Mas deve ser considerado o fato de que corresponde a uma das primeiras propostas no sentido da análise orientada a objetos.

Outras técnicas textuais podem ser encontradas em propostas de Booch [BOO 83] que centra-se mais no projeto orientado a objetos para a linguagem *Ada*, e de Mattoso & Blum [MAT 88] que é menos rigorosa na análise do texto mas introduz conceitos adicionais para a generalização e especialização de classes. Também deve ser mencionado Abbott [ABO 83] que, embora não pertencendo ao paradigma de orientação a objetos, deu a base para todas estas técnicas.

2.2.2. As Técnicas Evolutivas

Com foi indicado anteriormente, estas técnicas podem ser divididas considerando o aspecto da linha central de modelagem que estendem. Sem dúvidas, esta é a categoria mais populosa por razões óbvias: as técnicas originais são todas amplamente conhecidas e provadas no desenvolvimento de sistemas de software, então parece natural tentar estende-las para a orientação a objetos.

2.2.2.1. As Técnicas Orientadas a Dados

Esta subcategoria representa todas aquelas técnicas que utilizam extensões semânticas de modelos de dados ou a denominada modelagem de informação. O modelo de dados mais amplamente utilizado, pela sua divulgação e caráter intuitivo, é modelo entidade-relacionamento estendido.

Como a técnica mais representativa nesta subcategoria pode ser indicada a *Análise Baseada em Objetos* (*Object-Oriented Analysis* ou *OOA*) de Coad & Yourdon [COA 92]. A técnica propõe cinco atividades principais que resultam em um modelo de multicamadas, onde cada camada é construída sobre a camada anterior. Estas atividades são:

1. *Localização de Classes-&-Objetos*: São procuradas estruturas, outros sistemas, dispositivos, coisas ou eventos, papéis, procedimentos operacionais, locais e unidades organizacionais como possíveis objetos (ou *Classes-&-Objetos*, na terminologia dos autores).

2. *Identificação de Estruturas*: São identificados relacionamentos entre as Classes-&-Objetos. Estes relacionamentos podem ser de generalização-especialização (ou *Gen-Espec*, na terminologia dos autores), composição ou agregação (ou *Todo-Parte*) e estruturas múltiplas que combinam as anteriores.

3. *Identificação de Assuntos*: Os assuntos são mecanismos que permitem subdividir o domínio do problema em subproblemas. Para isto são examinadas as Classes-&-Objetos de mais alto nível em cada estrutura

previamente identificadas para torna-las assuntos candidatos. Estes assuntos candidatos são refinados para minimizar as interdependências.

4. *Definição de Atributos*: Cada atributo captura um *conceito atômico* de um objeto. São associados os atributos às Classes-&-Objetos segundo as estruturas Gen-Espec (herança). Também são determinadas as cardinalidades (limites inferior e superior) nos relacionamentos entre as Classes-&-Objetos.

5. *Definição de Serviços*: São identificados os métodos (serviços) que cada objeto realiza, tanto para o seu próprio uso como no benefício de outras Classes-&-Objetos (conexões de mensagens).

As ferramentas fundamentais da técnica de Coad & Yourdon são os Diagramas de Classes-&-Objetos e os Diagramas de Serviços. Neste últimos também encontram-se os Diagramas de Estado do Objeto.

Os diagramas de Classes-&-Objetos são construídos incrementalmente em cinco camadas associadas às atividades acima. A figura 2.2 mostra um exemplo que representa a própria técnica de OOA, i.e. é um metamodelo. Na figura é possível reconhecer Classes-&-Objetos na forma de caixas com cantos arredondados, os relacionamentos com semicírculo mostram estruturas Gen-Espec, e o relacionamento com triângulos é do tipo Todo-Parte. Os assuntos são representados pelas bordas numeradas nos cantos que mostram classes relacionadas. As cardinalidades dos relacionamentos são indicados pelos números junto a cada Classe-&-Objeto. Existem também notações para conexões de ocorrência (arco simples) e conexões de mensagens (setas grossas).

Os Diagramas de Serviços são muito semelhantes aos diagramas tradicionais de fluxo de controle. Adicionalmente incluem os Diagramas de Estado do Objeto que correspondem a um tipo simplificado de diagrama de transição de estados. A figura 2.3 ilustra um exemplo de Diagrama de Estado do Objeto. Na figura as caixas representam estados e as setas sem nome transições.

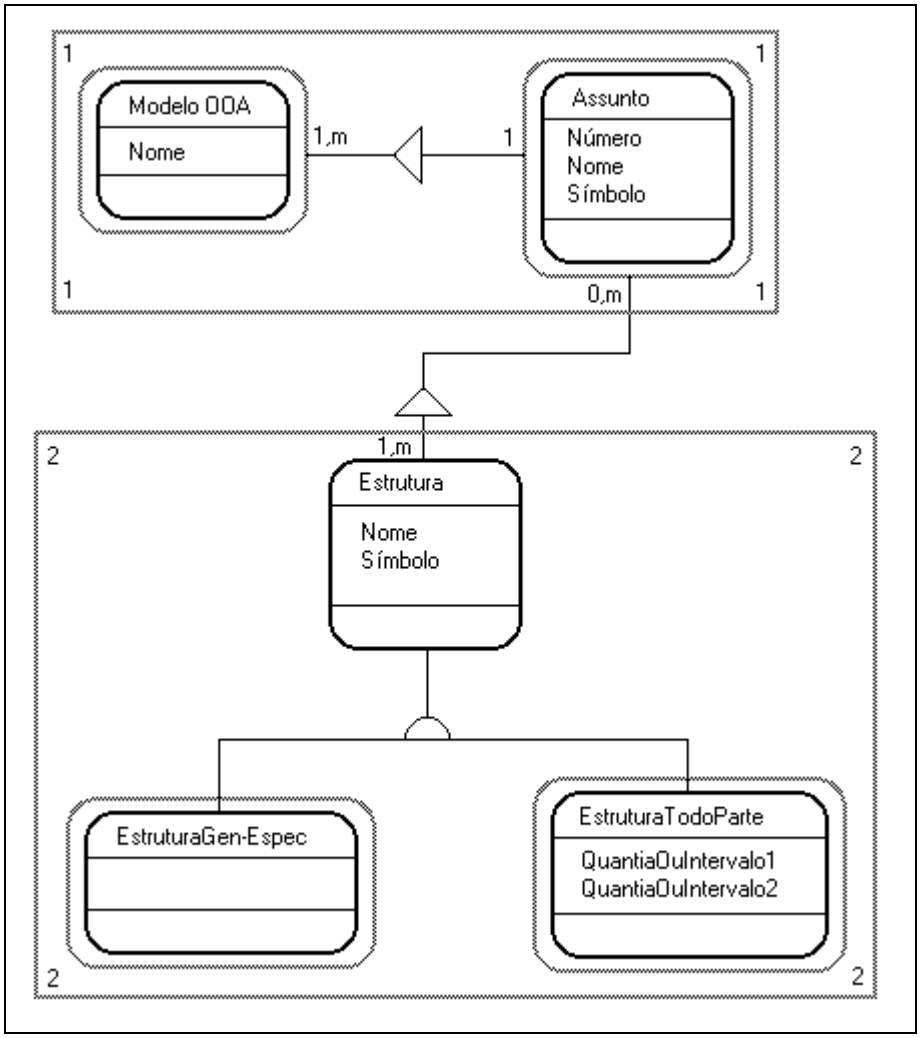

Figura 2.2 - Um subconjunto de um modelo OOA da OOA propriamente dita [COA 92].

A técnica centra-se fortemente na modelagem estrutural à qual dedica 4 capítulos no livro (do 3 ao 6) e somente dedica um para a modelagem dos serviços encapsulados e transições de estados (capítulo 7).

Posteriormente à divulgação da técnica têm sido propostas algumas extensões menores. Entre elas cabe destacar a de Schaschinger [SCH 92], denominada ESA (*Expert Supported OOA*), que propõe um passo específico para identificar o comportamento esperado do sistema (característica da categoria de técnicas comportamentais), adicionalmente a extensão é suportada por uma ferramenta CASE. Outra

contribuição é feita pelo próprio Coad [COA 92a] em que define *padrões* de agrupamento de classes que podem ser guias úteis para a identificação de assuntos.

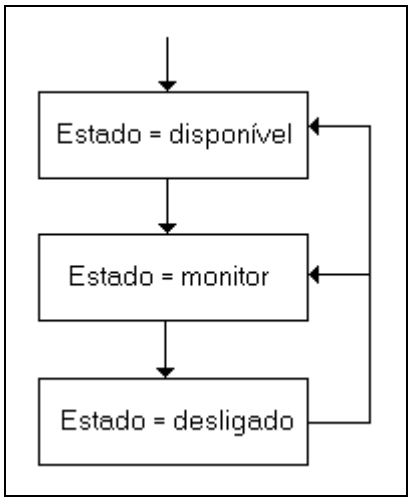

Figura 2.3 - Exemplo de um Diagrama de Estado da Classe-&-Objeto Sensor [COA 92].

Outra técnica que pode ser classificada nesta subcategoria é o *Modelo Entidade-Relacionamento Orientado a Objetos* (*OOER*) proposto por Navathe & Pillalamarri [NAV 89] que é mais uma proposta para bancos de dados orientados a objetos e não inclui outros aspectos de modelagem, porém é útil para a modelagem conceitual na AOO. Outra proposta semelhante à anterior é a de Manfredi et al. [MAN 89] que definem um procedimento apenas para a modelagem estrutural na AOO.

Em [KUR 89], Kurtz et al. propõem uma técnica que utiliza um modelo entidade-relacionamento estendido, inclusive com notações comuns, junto com uma rede de estados, baseada em rede de Petri restritas, para a modelagem de comportamento dos objetos[6]. Por sua parte, Shlaer & Mellor [SHL 90] apresentam o

[6] Esta técnica sofreu extensões posteriores (ver [CLY 92] e [EMB 92] e as técnicas integracionistas), visando melhorar a modelagem estrutural e comportamental, incluindo interações dinâmicas entre objetos.

que corresponde mais a uma técnica de modelagem de informação, já que não trabalha os aspectos funcionais (métodos) nem dinâmicos (estados e transições)[7].

Finalmente, Jacobson [JAC 87] e [JAC 92] propõe a técnica denominada *ObjectOry* (*Object-Oriented Systems Development Factory*) para todo o desenvolvimento orientado a objetos, mas que define passos explícitos para AOO: 1) modelagem de entidades, 2) modelagem de casos de usos (seqüências de transações como resultado de diálogos entre o sistema e os usuários), e 3) modelagem de serviços (para os casos de uso oferecidos pelas entidades). Esta proposta baseia-se em técnicas para desenvolver sistemas de grande porte (*block design*), modelagem conceitual e orientação a objetos.

2.2.2.2. As Técnicas Orientadas às Funções

Esta subcategoria representa todas aquelas técnicas que utilizam extensões de modelos funcionais com decomposição funcional. O modelo funcional mais amplamente utilizado, também pela sua divulgação e caráter intuitivo, é o diagrama de fluxo de dados (*DFD*).

Como a técnica mais representativa nesta subcategoria pode ser indicada o *método para a especificação de requisitos orientado a objetos* (*Object-Oriented Requirements Specification Method*) de Bailin [BAI 89]. Este método de AOO baseia-se nos Diagramas de Fluxo de Dados de Entidades (*Entity Data Flow Diagrams*) que representam tanto *entidades* (objetos) como *funções* (métodos).

O método propõe os seguintes passos:

1. *Identificar as entidades (objetos) chaves no domínio do problema*: Construir um DFD tradicional identificando entidades candidatas nos objetos dos nomes dos processos (os nomes dos processos assumem a forma: ação + objeto). Construir um diagrama entidade-relacionamento (DER) e um dicionário de dados com estas entidades.

[7] Posteriormente, esta técnica viria a ser estendida para estes aspectos pelos mesmos autores. Ver as técnicas integracionistas e as referências [SHL 89] e [SHL 92].

2. *Distinguir entre entidades ativas e passivas*: As entidades ativas devem ser aquelas que aparecem como processos e cujas funções (métodos) são consideradas na fase de análise. As entidades passivas são aquelas que aparecem como fluxos de dados e suas funções são consideradas na fase de projeto.

3. *Estabelecer fluxos de dados entre as entidades ativas*: Construir um Diagrama de Fluxo de Dados de Entidades (DFDE) de mais alto nível que contém somente entidades. As entidades ativas do DER do passo 1 devem tornar-se processos no DFDE e as entidades passivas devem ser fluxos ou depósitos de dados. Toda entidade do DER deve estar no DFDE e vice-versa.

4. *Decompor entidades (ou funções) em subentidades e/ou funções*: Explodir o DFDE inicial segundo as regras: 1) uma entidade pode ser decomposta em subentidades e/ou funções, 2) cada subentidade deve possuir um DFDE, 3) as subentidades compõem a entidade superior, 4) as funções (métodos) são realizados pela entidade superior, e 5) uma função só pode ser decomposta em subfunções.

5. *Procurar novas entidades*: Considerar em cada decomposição se as novas funções implicam novas entidades. Incluir as novas entidades nos DFDE com os fluxos e depósitos.

6. *Agrupar as funções sob as novas entidades*: Identificar todas as funções realizadas por ou sobre as novas entidades. Mudar entidades passivas para ativas se for necessário e reorganizar os DFDE's.

7. *Definir domínios apropriados para as entidades*: Para lidar com a complexidade cada entidade deve pertencer a um domínio (por exemplo domínios de aplicação, domínios da tecnologia, domínios da ciência da computação, modelos de execução e domínios de execução). Construir um DER separado para cada domínio.

A técnica resulta em um DER (documentado com um dicionário de dados), que utiliza a mesma notação tradicional, junto com uma hierarquia de DFDE.

Ainda que a técnica de Bailin suporte orientação a objetos, não são usados explicitamente os conceitos deste paradigma.

É importante destacar que a proposta não faz uma decomposição funcional pura, mas particiona entidades (objetos) em funções e subentidades, e as funções em subfunções. Assim as funções sempre pertencem a alguma entidade de mais alto nível.

As figuras 2.4 e 2.5 mostram exemplos de DFDE's. As entidades (passivas ou ativas) são distinguidas das funções indicando o nome entre colchetes []. Nas figuras, *[Cliente]*, *[Conta]*, *[Transações]*, *[Estados de Conta]* e *[Cheques]* são entidades. Os outros elementos desempenham o mesmo papel que nos DFD's tradicionais.

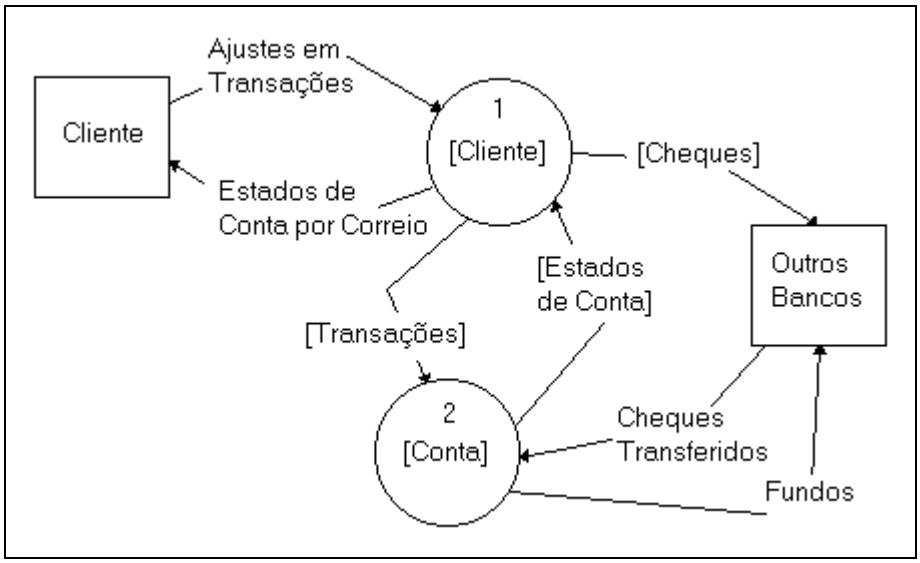

Figura 2.4 - Exemplo específico de um Diagrama de Fluxo de Dados de Entidades (DFDE) [BAI 89].

Adicionalmente ao procedimento e os modelos indicados, Bailin sugere algumas extensões à técnica, que não são desenvolvidas na proposta. Entre elas cabe destacar a definição de serviços *end-to-end*, o enriquecimento do modelo entidade-relacionamento com tipos de relacionamentos pré-definidos, a definição explícita das funções como atributos das entidades, os relacionamentos entre domínios e um modelo

de estados e transições (reconhecendo a dificuldade para correlacionar os estados e as transições deste modelo com o DFDE).

Outra proposta que pode ser classificada como técnica evolutiva orientada à função é a de Girardi & Price [GIR 90], que define listas de entidades e de tarefas, decompondo estas últimas para identificar novas classes ou associar as tarefas aos objetos.

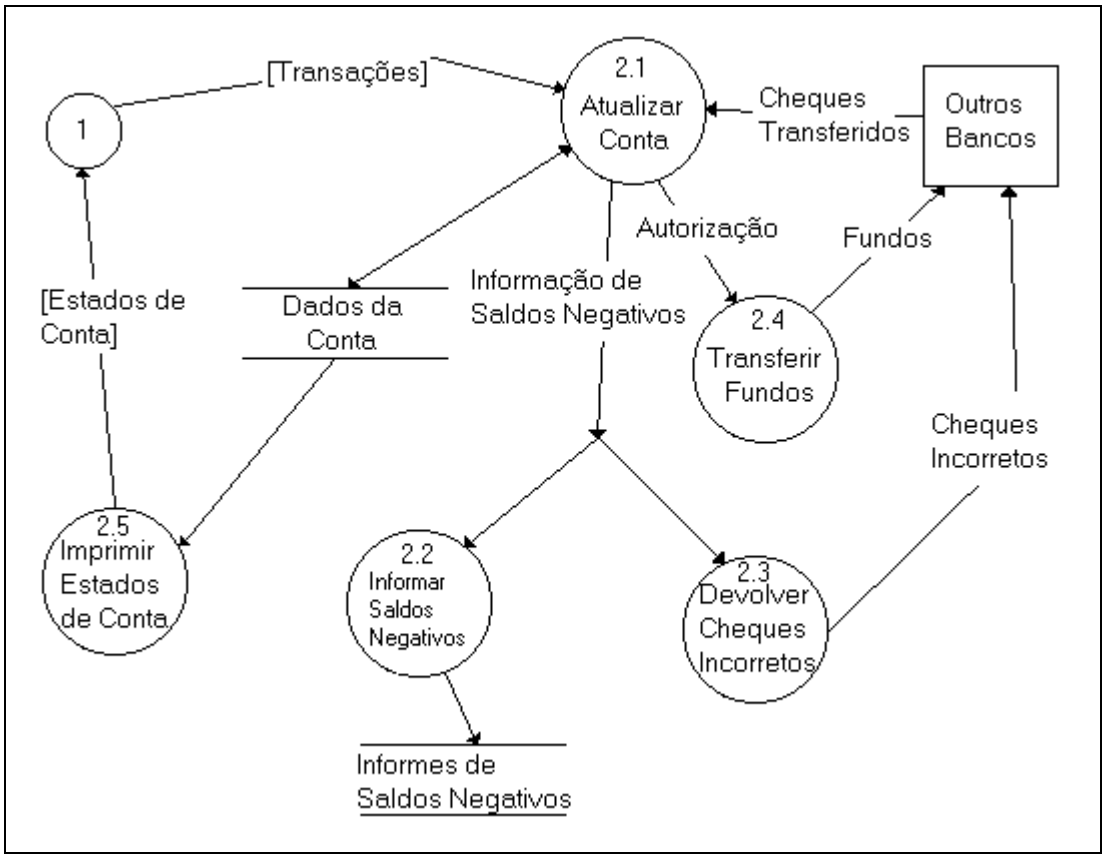

Figura 2.5 - Exemplo específico de um DFDE da decomposição da entidade *[Conta]* da figura 2.4 [BAI 89].

Outra técnica mais orientada ao projeto, mas que trata da análise, é a de Booch [BOO 86], que propõe construir um DFD de alto nível, identificando todas as fontes, destinos e depósitos de dados como objetos, incluindo o processo maior de transformação que também pode ser um objeto. Em [SEI 89], Seidewitz também propõe

construir um DFD para gerar um grafo de entidades que é mapeado em um diagrama de objetos. Já Alabiso [ALA 88], propõe explicitamente a transformação de modelos de fluxo de dados da análise para o projeto orientado a objetos, usando como base as extensões aos DFD de Ward & Mellor [WAR 85]. Por sua parte, Bulman [BUL 89] recomenda centrar o interesse da AOO nos depósitos e entidades externas modeladas nos DFD's, e complementar este modelo com diagramas entidade-relacionamento. O autor também acrescenta passos para o POO. Finalmente, Lee & Carver [LEE 91] propõem abstrair fatos, no sentido usado na modelagem de conhecimento, dos DFD's, para derivar objetos e ações.

2.2.2.3. As Técnicas Orientadas à Dinâmica

Esta subcategoria representa todas aquelas técnicas que utilizam extensões de modelos dinâmicos de alguma espécie. Os modelos dinâmicos mais utilizados são os diagramas de transição de estados, os *statecharts* e as redes de Petri.

Como a técnica mais representativa nesta subcategoria pode ser indicada a proposta por Kappel, cuja referência é [KAP 91], mas que apenas será descrita a partir de [MON 92]. Esta proposta é uma versão orientada a objetos dos diagramas de transição de estados e as redes predicado/transição.

A representação utilizada pela proposta de Kappel inclui notações para a visão estática de objetos, atributos, comportamento e relacionamentos de generalização, agregação e outros. Para a visão dinâmica, possui notações para a comunicação (troca de mensagens) de objetos, controle (seqüência de eventos) e restrições para o comportamento dinâmico.

Outra técnica classificável como evolutiva orientada à dinâmica é a de Schiel & Mistrik [SCH 90]. A metodologia é denominada *OKAY* (*Object-oriented Knowledge Analysis and Design*) e consiste em 8 passos, dos quais os primeiros 4 correspondem à fase de análise dos requisitos da aplicação e ao desenvolvimento de um modelo inicial do sistema. Estes passos são: 1) desenvolvimento de um grafo de comportamento (rede hierárquica de fluxos de informação que inclui dinâmica), 2) derivação de um grafo de estrutura de informação (modelo estrutural de objetos), 3) análise temporal dos objetos (tipo *snapshot*, *rollback*, histórico e temporal), e 4) análise

de restrições (tempo de acesso, tabela de pré e pós-condições, tabela de regras). Por ser esta uma técnica pouco ortodoxa, não foi escolhida como a mais representativa desta categoria.

2.2.3. As Técnicas Integracionistas

Esta categoria representa àquelas técnicas que possuem vários modelos separados das diferentes dimensões.

Como técnica representativa desta categoria encontra-se a de Rumbaugh et al. [RUM 91]. Os autores propõem uma metodologia de desenvolvimento de software orientada a objetos denominada OMT (*Object Modeling Technique*), que inclui explicitamente a AOO.

A AOO na técnica OMT, consiste na construção de três modelos, um para cada dimensão, que especifiquem o domínio do problema considerando os requisitos. O procedimento da AOO está intimamente ligado à construção de modelos destes três aspectos: modelagem de objetos, modelagem dinâmica e modelagem funcional.

- *Modelagem de Objetos*: Consiste na construção de um modelo que representa o aspecto estático do sistema. Os passos desta modelagem são:

  1. identificar objetos e classes;
  2. preparar um dicionário de dados;
  3. identificar associações (incluindo agregações ou composições) entre os objetos;
  4. identificar atributos dos objetos e associações;
  5. organizar e simplificar as classes de objetos usando herança;
  6. verificar os passos de acesso para consultas;
  7. iterar e refinar o modelo; e
  8. agrupar as classes em *módulos*.

  Esta modelagem, que já tinha sido apresentada em [LOO 87], é fortemente influenciada por modelos do tipo entidade-relacionamento, porém apresentado uma notação própria pouco ortodoxa. A figura 2.6

mostra um exemplo de um modelo de objetos. Na figura, as caixas representam objetos com nome e atributos, e as associações (que podem possuir atributos como no exemplo) são representados por arcos. A cardinalidade é indicada pelo símbolo (•), que em este caso indica que 0 ou mais ocorrências participam da associação *Trabalha_para*. Adicionalmente, existem notações para generalização, agregação ou composição, ordenamento de instâncias, associações ternárias, instanciação, herança múltipla, atributos derivados e restrições.

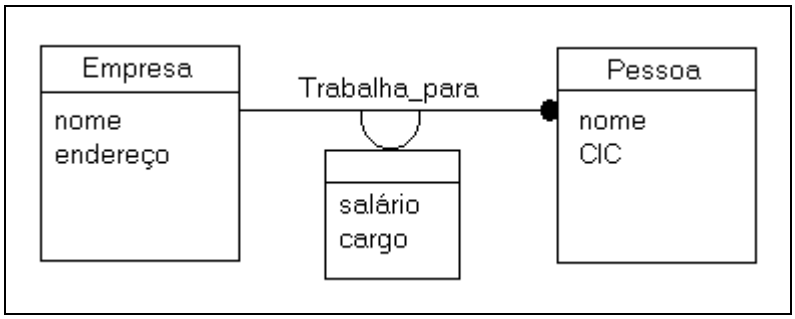

Figura 2.6 - Exemplo específico de um Modelo de Objetos [RUM 91].

- *Modelagem Dinâmica*: Consiste na construção de modelos que representam o comportamento dinâmico da interação dos objetos em termos de estados, transições, eventos e ações. Os passos para esta modelagem são:

  1. preparar cenários de seqüências de interações típicas;
  2. identificar eventos entre objetos;
  3. preparar um seguimento de eventos para cada cenário;
  4. construir um diagrama de estado; e
  5. verificar os eventos entre os objetos para verificar consistência.

  O modelo construído está baseado nos *statecharts*, e também tinha sido proposto anteriormente em [BEA 90]. Adicionalmente, em [WAL 92] sugere-se que os *statecharts* possam ser usados para mecanismos padronizados de colaboração entre classes de objetos (do tipo *diretor*, *agente* e *servidor*).

As ações do modelo dinâmico mais complexas do ponto de vista computacional, devem ser acrescentadas ao modelo de objetos.

A figura 2.7 mostra um exemplo do Diagrama de Estados de Rumbaugh et al. para um exemplo específico. Na figura, os estados de cada classe são representados por caixas de cantos arredondados. As transições têm associados os eventos que as provocam. Todos os estados que compõem um superestado (como *Disponível* e *Ocupado* no superestado *Recurso* da classe do mesmo nome) são disjuntos. No exemplo pode observar-se a sincronização de transições na ocorrência dos eventos: *Peça usa recurso* e *Peça libera recurso*. Adicionalmente, são previstas notações para ações nas transições e nos estados, transições condicionadas, geração de eventos nas transições, generalização de estados, e subdiagramas concorrentes.

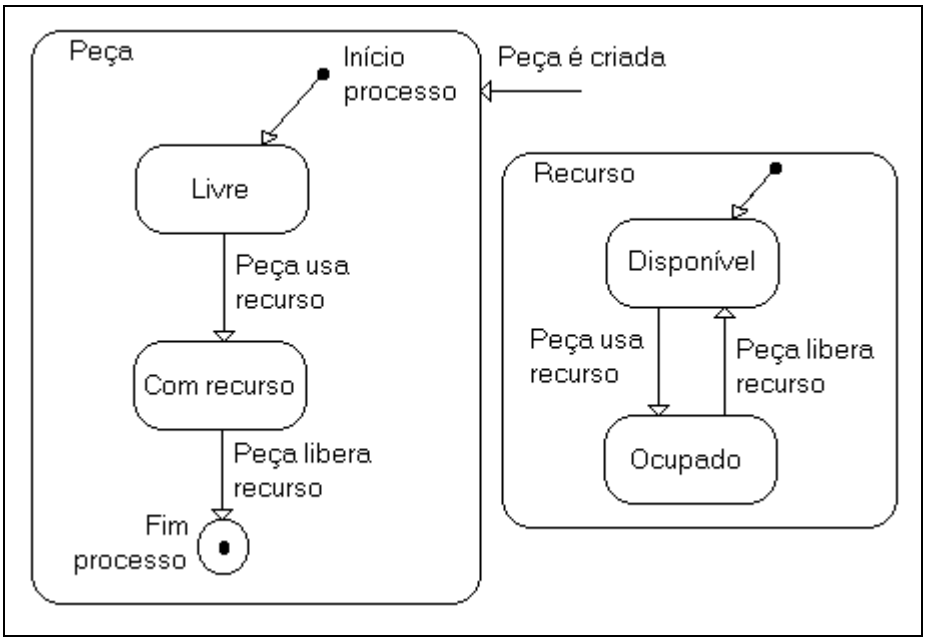

Figura 2.7 - Exemplo específico de um Diagrama de Estados do Modelo Dinâmico [RUM 91].

- *Modelagem Funcional*: Consiste na construção de um modelo funcional baseado em DFD's para representar as transformações ao interior do sistema. Os passos que devem ser realizados para esta modelagem são:

    1. identificar valores de entrada e saída;
    2. construir um diagrama de fluxo de dados que mostre as dependências funcionais;
    3. descrever as funções;
    4. identificar as restrições; e
    5. especificar critérios de otimização.

    Esta modelagem funcional descreve as funções sem associação explícita com os objetos. Contudo, algumas funções básicas (do último nível de decomposição dos DFD's) de interesse computacional podem ser indicadas no modelo de objetos.

    A figura 2.8 mostra um exemplo específico de DFD como utilizado na técnica OMT. Na figura, as elipses representam funções, e os arcos podem representar atributos (por exemplo *nome*) ou objetos (por exemplo *Contas*). Os depósitos de dados correspondem a objetos (*Banco* e *Conta*), o mesmo acontece com as entidades externas (*Cliente* na figura). A seta com a ponta maior representa a criação de uma instância de objeto na classe *Conta*. Outras notações indicam fluxos de controle (linhas pontilhadas) e composição, decomposição e duplicação de fluxos de dados.

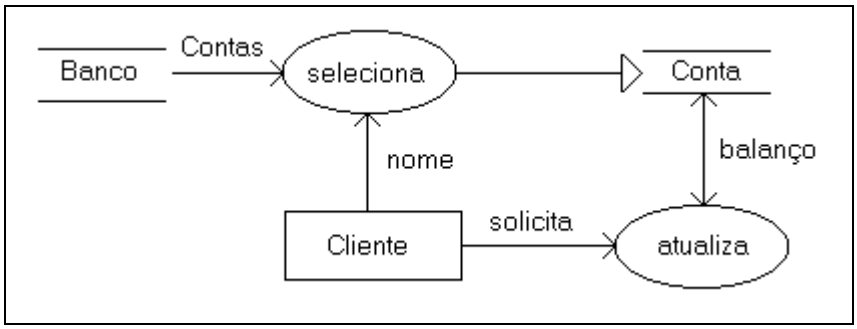

Figura 2.8 - Exemplo específico de um Diagrama de Fluxo de Dados do Modelo Funcional [RUM 91].

A técnica OMT tem se constituído em uma das técnicas mais divulgadas, existindo diferentes ferramentas que incluem sua notação. Também tem sido objeto de avaliações, como por exemplo a de Hayes & Coleman [HAY 91] que busca formalizar e precisar a semântica dos modelos utilizados na técnica, e a de Bruegge et al. [BRU 92] que avalia sua aplicação em um projeto universitário.

Posterior à modelagem nas três dimensões, Rumbaugh et al. sugerem iterar e acrescentar operações (métodos, ações e funções) onde for necessário.

A integração final destes modelos, por exemplo a associação das funções do modelo funcional e as ações do modelo dinâmico aos objetos, é feita na fase posterior denominada *Projeto de Objetos*.

Outra técnica nesta mesma categoria é a de Shlaer & Mellor [SHL 89] e [SHL 92], que estendem a proposta inicial deles mesmos apresentada em [SHL 90], referenciada neste trabalho como técnica evolutiva orientada a dados. Os autores acrescentam ao desenvolvimento do modelo de informação, a definição de ciclos de vida dos objetos, definição de relacionamento dinâmicos, modelagem de processos e definição de domínios e subsistemas. Com isto, a especificação consiste em mais de 10 componentes, entre modelos e diagramas, que descrevem as três dimensões: informação, dinâmica e processo.

Também pode ser considerada como técnica integracionista a proposta por Clyde et al. em [CLY 92] e [EMB 92], que estende a técnica inicial de [KUR 89]. Os autores definem três modelos: 1) Modelo Objeto-Relacionamento (*Object-Relationship Model* ou *OBM*) que descreve os objetos e seus relacionamentos estáticos com um modelo semântico tipo entidade-relacionamento, 2) Modelo Objeto-Comportamento (*Object-Behavior Model* ou *OBM*) que descreve o comportamento dos objetos de cada classe usando redes de Petri modificadas, e 3) Modelo Objeto-Interação (*Object-Interaction Model* ou *OIM*) que descreve as interações e vários tipos de restrições, incluindo funcionais e temporais. Uma consideração interessante desta técnica é que está definida em termos informais e formais.

2.2.4. As Técnicas Reversas

As técnicas reversas podem ser muitas vezes confundidas com outras categorias, e isto não é incomum porque no momento de sustentar os conceitos de uma linguagem de programação orientada a objetos pode ser apropriado utilizar enfoques, notações e procedimentos de diversa natureza.

A técnica escolhida como mais representativa (e mais recente) é a proposta por Nerson [NER 92], que define uma técnica de análise e projeto orientados a objetos para a linguagem *Eiffel* [MEY 88], usando a notação de objetos melhorada (*Better Object Notation* ou *BON*) [NER 91].

Esta técnica consiste em três passos:

1. *Identificação, nomeação e "clustering" de classes*: São identificadas as classes que possuem estados internos e oferecem serviços. As classes encontradas devem ser agrupadas em *clusters*, incluso o particionamento inicial pode começar com um *cluster*. Neste passo são gerados diagramas de *cluster* (*cluster chart*) que listam e definem as classes.

2. *Identificação de eventos e protocolos de comunicação entre objetos*: Os eventos são classificados em eventos externos (gerados desde o exterior do sistema) e internos (comportamentos de partes do sistema). Os eventos internos são mapeados em mensagens que definem os protocolos de comunicação de objetos. São criados os diagramas de eventos por *clusters*.

3. *Definição de classes e projeto preliminar da arquitetura básica*: A partir dos diagramas de *cluster* são definidos os diagramas de classes (*class chart*) que mostra informação classificada em *questões* (informação para outras classes), *comandos* (serviços requeridos por outras classes) e *restrições* (conhecimento que a classe deve manter). São formalizadas a estrutura e o comportamento do sistema em um modelo estático e um modelo dinâmico, respectivamente. Os relacionamentos possíveis no

modelo estático são de herança e cliente-fornecedor (associação e agregação). O modelo dinâmico mostra os protocolos de comunicação.

A técnica reconhece seus fundamentos em conceitos da linguagem *Eiffel* [MEY 88] (conceito de *cluster*, relacionamento cliente/fornecedor, classes diferidas e modelos estático e dinâmico), a técnica *OBA* (apresentada na próxima seção) e os conceitos de cartões de classes de [BEC 89].

É importante destacar que as decisões sobre relacionar classes por derivação, associação, generalização ou especialização são feitas no projeto e não na análise, segundo Nerson. Os passos do POO, segundo [NER 92], seriam: 1) descrição, indexação e instanciação de classes; e 2) abstração e classificação.

As tabelas da 2.3 à 2.6 mostram alguns exemplos da notação *BON* da técnica de Nerson para um sistema de locação de veículos.

DIAGRAMA DE CLUSTER: *LOCADORA DE VEÍCULOS*	
CLASSE	**DEFINIÇÃO**
CLIENTE	Locatário, individual ou empresa
CONTRATO	Termos da locação com as condições de pagamento
LOCAÇÃO	Informação da locação quando devolvido o veículo
VEÍCULO	Automóvel selecionado para a locação
MODELO	Descrição das características
TAXA	Condições dos preços

Tabela 2.3 - Exemplo específico de um Diagrama de *Cluster* para um sistema de locação de veículos [NER 92].

Diagrama de Eventos		
DOMÍNIO DE APLICAÇÃO	**EVENTO EXTERNO**	**EVENTO INTERNO**

Companhia de locação de veículos	*Cliente faz reserva de um carro.* *Um carro locado quebra.* *Um carro novo chega.* *Um carro é devolvido.*	*Contrato de locação é impresso na devolução do carro.* *Abastecer o carro na devolução.* *Verificar a quilometragem na devolução.* *Inspeção do carro.*
Controlador de tráfego	*Um veículo chega na interseção.* *Um veículo deixa a interseção.* *Uma rua está fechada ao tráfego.*	*Abre o sinal do semáforo.* *Fecha o sinal do semáforo.*

Tabela 2.4 - Exemplo específico de um diagrama de eventos para domínios do sistema de locação de veículos [NER 92].

TIPOS DE *CLASSES* DO RELACIONAMENTO *CLASSE*			
relacionamento de herança	*descendente* é-um *pai* *descendente* comporta-se como *pai* *descendente* implementa *pai* *descendente* combina com *pai* *pai* difere-para *descendente* *pai* fatoriza *descendente*	relacionamento cliente/fornecedor	*cliente* usa *fornecedores* *cliente* necessita *fornecedores* *cliente* possui-um *fornecedor* *cliente* consiste-em *fornecedores* *fornecedor* fornece-a *clientes*

Tabela 2.5 - Semântica dos relacionamento de herança e cliente/fornecedor no modelo orientado a objetos de Nerson [NER 92].

VEÍCULO		
Nome do *Cluster*: *PROPRIEDADES DA LOCAÇÃO*		
TIPO DE OBJETO: Automóvel selecionado para a locação		**COMPORTA-SE COMO**: *ITEM_LOCADO*
Questões	**Comandos**	**Restrições**
Modelo do carro. *Placa.* *Disponibilidade.* *Lugar da locação.* *Lugar da devolução.*	*Verificar quilometragem.* *Abastecer o carro.* *Trocar o óleo.*	*Lugares da locação e devolução são os mesmos.*

Tabela 2.6 - Exemplo específico de um Diagrama de Classe para a classe *veículo* [NER 92].

A figura 2.9 mostra um modelo estático do *cluster transporte*. Os *clusters* são representados por uma caixa com cantos arredondados com o nome junto. As elipses representam classes. Os símbolos opcionais no topo de cada classe representam: (*) classes diferidas, e (+) classes descendentes não diferidas. As classes sublinhadas são classes reusadas. As notações para os relacionamentos são os seguintes: herança seta simples do descendente ao pai (no exemplo *USUÁRIO* a *PESSOA*), cliente/fornecedor usa dupla linha com seta (associação de *VEÍCULO* e *USUÁRIO*) e dupla linha com base aberta (agregação ou composição de *MOTOR* em relação a *CARRO*).

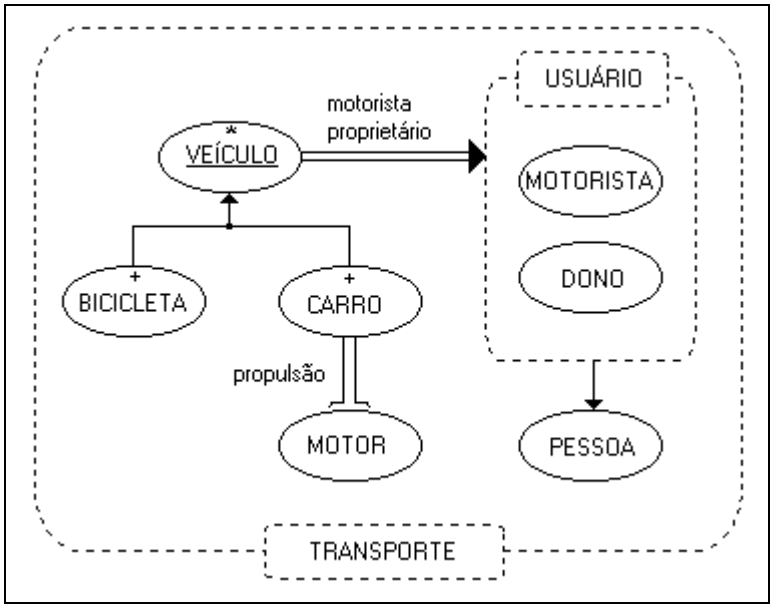

Figura 2.9 - Modelo estático para o *cluster transporte* [NER 92].

A figura 2.10 mostra um modelo dinâmico, onde as caixas representam objetos e as caixas sombreadas múltiplas instâncias. Os protocolos de comunicação são representados com linhas numeradas. Os números são comentados em cenários.

Outra técnica que pode ser classificada em esta categoria é a proposta mais recente de Booch [BOO 91], que tem evoluído desde um método específico de projeto para a linguagem *Ada* até uma proposta mais geral, porém ainda com conceitos daquela linguagem. Os grandes passos desta técnica são: 1) identificar classes e objetos,

2) identificar a semântica das classes e objetos, 3) identificar relacionamentos entre as classes e objetos, e 4) implementar classes e objetos. As ferramentas utilizadas são diagramas e *templates* para classes, objetos, processos e módulos, *templates* de operações, diagramas de transição de estados e diagrama de temporização (*timing diagram*).

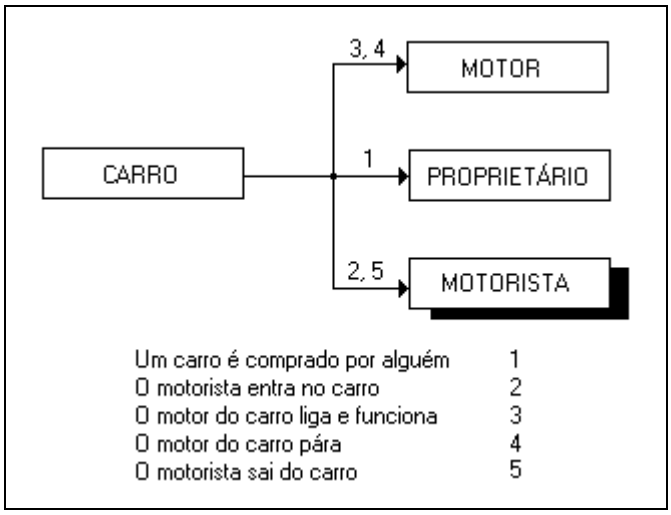

Figura 2.10 - Exemplo específico de um Diagrama de *Cluster* para um sistema de locação de veículos [NER 92].

Também na linha da linguagem *Ada*, encontra-se a proposta de *Análise de Requisitos Orientada a Objetos* de Freitas et al. [FRE 90]. Os passos propostos são: 1) listar as referências para possíveis objetos e classes, 2) eliminar valores, métodos, duplicações e outros não-objetos, 3) selecionar objetos e classes, 4) agrupar os objetos em classes, e 5) selecionar requisitos para cada objeto ou classe. Finalmente, e terminando com *Ada*, Ladden [LAD 88] propõe os seguinte passos para AOO na sua metodologia de desenvolvimento orientada a objetos: 1) definir todos os objetos do espaço do problema, 2) definir todas as primitivas funcionais associadas com cada objeto (métodos) do espaço do problema, e 3) definir cada requisito contratual satisfeito pelos objetos do espaço do problema e suas primitivas funcionais associadas (usando *inglês estruturado* e/ou equações).

Uma proposta que toma uma perspectiva diferente é a do grupo do *Senter for Industriforskning* (*SI*) [WIR 90] que propõe uma técnica denominada *Análise de Pa-*

péis, Síntese e Estruturação Orientadas a Objetos (*Object-Oriented Role Analysis, Synthesis and Structuring* ou *OORASS*), de desenvolvimento com orientação a objetos para a linguagem *Smalltalk-80*. Esta técnica não distingue uma fase análise, porém define alguns de seus passos nos seguintes termos: 1) Modelagem de Papéis: separação do domínio do problema em áreas de interesse (que podem ter sobreposição), cada área é modelada considerando uma estrutura de objetos que interagem desempenhando papéis, estes modelos são integrados e *sintetizados* em um modelo maior; e 2) Especificação dos Objetos: são especificados os objetos e seus papéis a nível de unidade, considerando seus colaboradores imediatos. São construídos diagramas de modelo de papéis das áreas de interesse, uma árvore de integração dos modelos de papéis e as especificações dos objetos. Os passos seguintes tratam com a implementação das classes (programas em *Smalltalk*), especificação da estrutura e a instanciação de objetos.

2.2.5. As Técnicas Comportamentais

As técnicas comportamentais têm sua origem na estratégia diferente para a identificação dos objetos: eles são derivados a partir do comportamento externo que deve exibir o sistema.

A técnica mais representativa é a *Análise do Comportamento de Objetos* (*Object Behavior Analysis* ou *OBA*) proposta por Rubin & Goldberg [RUB 92], mas que tinha um antecedente em [GIB 90]. O comportamento esperado do sistema é o ponto de partida: permite identificar os objetos que iniciam e participam do comportamento.

A proposta consiste em cinco passos iterativos:

1. *Definir o contexto da análise*: Este passo normalmente não corresponde às técnicas de análise, mas os autores o incluem. Trata com metas e objetivos, recursos, áreas de atividades e plano preliminar de análise.

2. *Entender o problema*: Consiste na definição de cenários de comportamentos para derivar *scripts* de comportamentos. Os *scripts* descrevem iniciadores, ações, participantes e serviços de cada comportamento dos sistema. Adicionalmente são desenvolvidos glossários de *parties*

(iniciadores e participantes) e serviços. Finalmente, são derivados os atributos de cada *party* e criado um glossário para eles.

3. *Definir os objetos*: São potenciais objetos todos os iniciadores que também são participantes, os participantes que não são iniciadores, e os participantes que são iniciadores que não possuem uma sobrecarga de papéis/responsabilidades. Definir os *Cartões de Modelagem de Objeto* (*Object Modeling Cards*) que incluem os atributos do objeto, os serviços do objeto fornecidos a outros objetos e os serviços do objeto contratados de outros objetos.

4. *Classificar os objetos e identificar os relacionamentos*: São definidos os relacionamentos contratuais dos objetos através dos serviços fornecidos/contratados. Os objetos são ordenados hierarquicamente de acordo com os conceitos de abstração, generalização, especialização e fatorização.

5. *Modelar a dinâmica do sistema*: São gerados glossários com a definição dos estados de cada objeto. Identificam-se os eventos que produzem transições entre os estados, organizando os estados, transições e eventos em ciclos de vida do objeto. Finalmente, são definidas as seqüências de operações através do ciclo de vida. Os autores recomendam usar *statecharts* [HAR 87] (mostrados nas técnicas integracionistas quando da descrição de técnica OMT [RUM 91]) para representar os ciclos de vida.

A técnica faz mais uso tabelas do que de diagramas: tabelas para *scripts*, aliases, glossários de *parties*, serviços e atributos. Além das tabelas, existem os cartões de modelagem de objetos, derivados dos cartões propostos em [BEC 89], e os diagramas de relacionamentos contratuais e organizacionais, e os *statecharts*. A especificação do sistema com OBA fornece então:

- *Scripts* que cadastram o comportamento ou o uso do sistema (proposto)

- Glossários de nomes de iniciadores/participantes, serviços dos participantes, atributos e definições de estados

- Modelos de objetos que representam os relacionamentos hierárquicos e contratuais
- Modelos da dinâmica do sistema que expressam os ciclos de vida dos objetos e as seqüências de operações

As tabelas 2.7 e 2.8 mostram a notação das tabelas de *script* e glossário de atributos, respectivamente.

Iniciador	**Ação**	**Participante**	**Serviço**
coisa1	notifica	coisa2	coisa2 pode ser notificada
coisa1	fornece informação a	coisa2	coisa2 pode aceitar informação
coisa1	requer informação de	coisa2	coisa2 pode prover informação
coisa1	requer serviço de	coisa2	coisa2 pode prover serviço

Tabela 2.7 - Notação de uma tabela de *script* [RUB 92].

Glossário de Atributos - Empregado							
Nome	**Definição**	**Contrato**	**Acessador**	**Atualizador**	**Multi/Mono Valorado**	**Valores**	**Definição de Estado**
código	código da empresa para o empregado	nenhum	nenhum	modifica código	mono	99999	não
salário	salário do empregado	nenhum	nenhum	modifica salário	mono	$99999,99	não

Tabela 2.8 - Exemplo específico de um glossário de atributos para o objeto *Empregado* [RUB 92].

A figura 2.11 mostra um exemplo de um cartão de modelagem de objeto.

A outra técnica, que não difere muito desta, e que é também denominada *Análise do Comportamento de Objetos* (*OBA*), é descrita em [GIB 90]. Corresponde a uma proposta menos elaborada, mas consistente em si mesma. A principal diferença radica em que não possui modelagem dinâmica, os termos utilizados variam um pouco: os iniciadores são *agentes* e os participantes são *receptores*. Adicionalmente, propõe modelar processos para poder *simular* o comportamento do sistema através das seqüências destes processos.

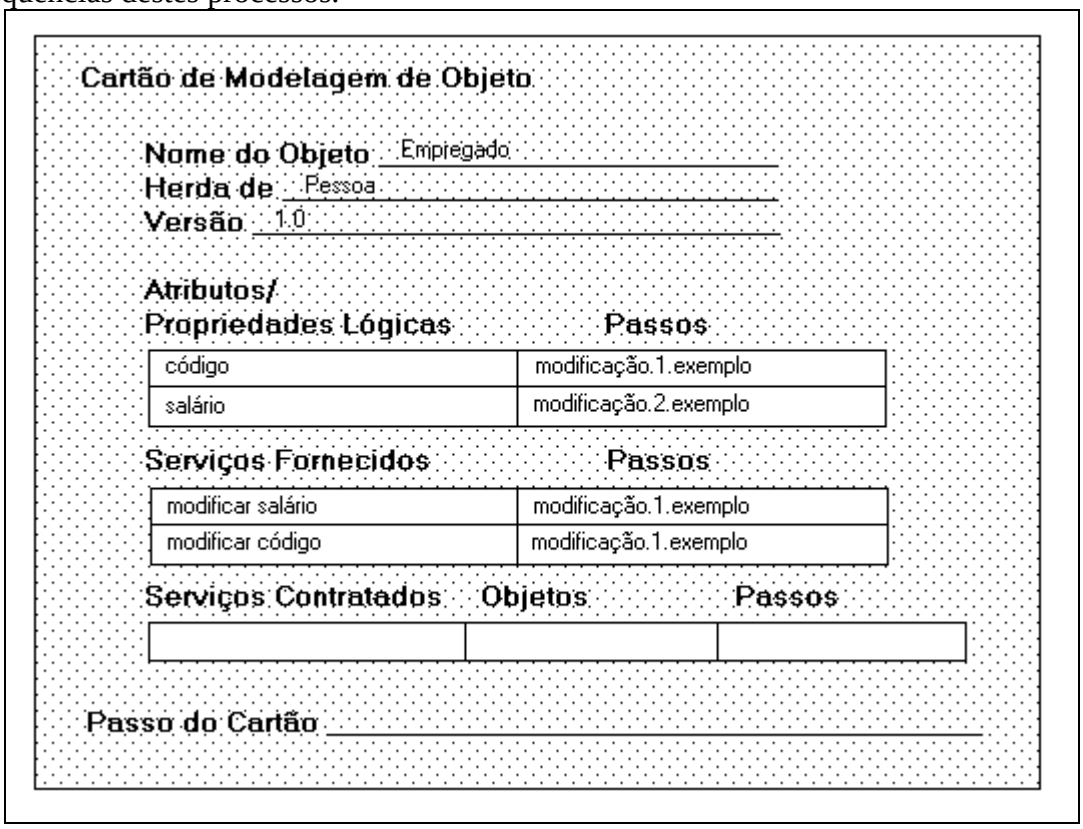

Figura 2.11 - Exemplo específico de um cartão de modelagem de objeto para *Empregado* [RUB 92].

Finalmente, uma técnica vinculada a esta categoria mas que está mais centrada no projeto do que a análise é a denominada *Projeto Dirigido por Responsabilidades* (*Responsability-Driven Design*) de Wirfs-Brock et al. [WIR 90] e [WIR 90a]. Esta proposta define os seguintes passos: 1) identificar as classes, 2) identificar responsabilidades e associa-las às classes, 3) identificar colaborações, 4) definir hierarquias, 5) definir subsistemas, e 6) definir protocolos. Os primeiros passos são claramente associados à AOO. Esta técnica utiliza também cartões e especificações

de classes, além de grafos de colaboração, diagramas de hierarquia, cartões e especificação de subsistemas e diagramas de Venn.

Desta forma completa-se a descrição da técnicas pertencentes a cada categoria. O próximo capítulo mostra os critérios que serão utilizados para realizar a comparação.

3. DEFINIÇÃO DE CRITÉRIOS DE COMPARAÇÃO

Os critérios que serão utilizados para realizar a comparação das técnicas de análise orientada a objetos, foram parcialmente definidos a partir de critérios semelhantes apresentados em [MON 92] e [FIC 92], e em menor grau, de adaptações dos critérios mostrados em [YAD 88], [DAV 88] e [PRE 87].

Os critérios escolhidos podem ser divididos inicialmente em duas categorias:

- aqueles relativos ao processo de análise propriamente dito, e
- aqueles relativos à especificação e representação da análise.

3.1. OS CRITÉRIOS DE COMPARAÇÃO PARA O PROCESSO DE ANÁLISE

Os critérios de comparação desta categoria estão associados com a modelagem realizada durante o processo de análise. Para melhor realizar a comparação, a modelagem foi dividida de acordo com os aspectos ou dimensões na construção dos modelos. Assim, estes critérios são categorizados em:

- aqueles relativos à modelagem das dimensões, e
- aqueles vinculados com a integração das dimensões.

3.1.1. Os Critérios de Comparação para a Modelagem das Dimensões

Como foi indicado anteriormente as dimensões da modelagem orientada a objetos são:

1. aspecto estrutural dos objetos,
2. aspecto dinâmico do comportamento, e
3. aspecto funcional dos requisitos.

Os critérios para estas dimensões são os mesmos, e são classificados segundo a seguinte estrutura:

- identificação de componentes para cada dimensão,
- localização destes componentes exclusivamente na modelagem de cada dimensão, e
- particionamento da complexidade de cada dimensão.

3.1.1.1. Os Critérios de Comparação para a Dimensão Estrutural dos Objetos

No caso desta dimensão, deve ser feita a *identificação* dos seguintes componentes:

- C*lasses*: diz relação com a forma em que são identificadas as classes e os objetos existentes no domínio do problema. Esta identificação pode ser deixada integramente a critério do analista, ou basear-se em descrições em linguagem natural, ou ainda pode ser definido um procedimento para tal efeito.
- *Atributos*: relaciona-se com a forma em que são identificados os atributos dos objetos, e se são fornecidos critérios para distingui-los dos objetos.
- *Métodos*: diz relação com a forma em que identificam-se as operações privativas dos objetos.
- *Relacionamentos*: relaciona-se com a forma em que devem ser identificados os distintos tipos de relacionamentos entre as classes. Estes relacionamentos podem ser de generalização, de agregação, de composição e associações do tipo *cria*, *usa* ou outras específicas do domínio do problema.

A *localização* deve ser feita sobre os seguintes componentes:

- *Classes*: tem a ver com a localização de uma classe já identificada. Geralmente não existem procedimentos para realizar esta tarefa. Em [MEY

88] (que também é citado por [MON 92]), é mostrado um exemplo de um *editor de texto orientado a janelas*, esta classe pode ser alocada como uma subclasse da classe *janela*, ou da classe *editor de texto*, ou de ambas as classes (em este caso ter-se-ia um caso de herança múltipla), ou ainda de nenhuma delas. Em este último caso, a classe em questão pode ser um novo objeto composto por uma *janela* e um *editor de texto*.

- *Atributos*: diz relação com a forma em que é alocado um atributo em um objeto. Por exemplo, podem existir atributos vinculados a uma associação, nesta situação o atributo pode pertencer ao relacionamento, ou a uma das classes que participam do relacionamento.

- *Métodos*: deve responder à pergunta: a qual objeto pertence o método? Por exemplo, existe a denominada *Lei de Demeter* [WIR 90] e [SAK 88], que define regras para alocar métodos em objetos, porém parece mais apropriada para o projeto do que para a modelagem propriamente dita.

Finalmente, deve ser considerado na comparação o critério de *particionamento* que a técnica fornece para abordar a complexidade que apresenta a estrutura dos objetos. Isto permite enfrentar sistemas extremamente complexos utilizando grupos de trabalho que modelem separadamente as *partes* para logo serem integradas em um *todo*: o sistema.

3.1.1.2. Os Critérios de Comparação para a Dimensão Dinâmica do Comportamento

No caso desta dimensão, deve ser feita a *identificação* dos seguintes componentes:

- *Estados*: deve responder às perguntas: Quais os estados que apresentam os objetos? Quais os estados que apresenta o sistema? Esta dualidade objeto/sistema tenta considerar a visão *micro* e *macro* que algumas técnicas apresentam. Geralmente os estados estão determinados pelos domínios dos atributos, no caso dos objetos; ou por conjuntos de atributos no caso do sistema.

- *Transições*: diz relação com identificação das mudanças de estado, i.e. o passo de um valor específico de um atributo ou conjunto de atributos para um outro valor no domínio.

- *Eventos*: relaciona-se com forma em que é identificada a ocorrência de um fato que provoca uma ou mais transições.

- *Ações*: corresponde à identificação dos métodos dos objetos no contexto desta dimensão. Deve responder às perguntas: Quais as ações que ocorrem durante uma transição? Ou durante a permanência em um estado?

A *localização* deve ser feita considerando o seguinte componente:

- *Ações*: relaciona-se com a forma em que as ações são alocadas nas transições ou estados apropriados.

O critério de *particionamento* nesta dimensão relaciona-se com a forma em que pode ser particionado o sistema desde o ponto de vista do comportamento que apresenta. Este particionamento pode ser hierárquico, utilizando generalizações (super-estados) e especializações (sub-estados) de estados, ou ainda agrupando estados e/ou transições em domínios.

3.1.1.3. Os Critérios de Comparação para a Dimensão Funcional dos Requisitos

Para esta dimensão da modelagem deve ser feita a *identificação* dos seguintes componentes:

- *Funções*: corresponde à identificação dos métodos dos objetos no contexto desta dimensão. Dependendo da estratégia da técnica de análise, as funções podem ser identificadas livremente e serem depois alocadas nos objetos, ou as funções podem ser tomadas de entre os métodos já identificados, ou ainda é possível uma combinação de ambas as estratégias.

- *Fluxos de Dados*: diz relação com a identificação dos fluxos de dados como entradas e/ou saídas das diferentes funções.

No existe *localização* para esta dimensão.

Também deve ser considerado o critério de *particionamento* nesta dimensão. A complexidade funcional deve ser enfrentada de tal forma que permita identificar subfunções (subordinadas ou não a objetos) a partir de uma função mais global. O método mais comum é a estratégia *top-down*.

3.1.2. Os Critérios de Comparação para a Integração das Dimensões

Esta categoria de critérios de comparação preocupa-se com a forma em que são relacionadas ou integradas as dimensões modeladas na análise. Estes critérios são: a reusabilidade, a construção dos modelos, as verificações e o procedimento da análise.

3.1.2.1. O Critério de Comparação para a Reusabilidade

Este critério está relacionado com a forma em que é propiciada a reusabilidade. Esta que é considerada uma das principais vantagens da orientação a objetos deve ser considerada nos seus dois aspectos:

- reuso de componentes de diferente natureza, já armazenadas em uma biblioteca projetada para tal efeito, para a construção dos modelos, e
- fornecimento de novas componentes como produto da modelagem, a serem armazenadas nesta biblioteca para futuros reusos.

A reusabilidade pode ser um mecanismo de integração das dimensões, já que por exemplo as classes podem ser armazenadas integrando tanto seu aspecto estrutural como seu comportamento e funcionalidade. É possível ter também reuso somente das estruturas das classes (que é o reuso mais tradicional), ou ainda reuso de comportamento dos objetos e funcionalidade dos objetos em forma separada.

3.1.2.2. O Critério de Comparação para a Construção dos Modelos

Este critério tem a ver com a forma em que são construídos os modelos dos diferentes aspectos. Por exemplo, pode ser construído um modelo separado para cada dimensão, ou integrar duas dimensões em um modelo, ou ainda tentar integrar toda a multidimensionalidade em um único modelo.

3.1.2.3. Os Critérios de Comparação para a Verificação

A comparação aqui é feita considerando os critérios de verificação na integração das dimensões. Estas verificações podem ser de consistência, completeza e não-ambigüidade.

Tratando-se de uma modelagem multidimensional, a *consistência* entre os aspectos modelados é fundamental. Interessa, por tanto, que os componentes identificados em cada dimensão sejam consistentes com os componentes nas outras dimensões. Claramente a consistência possui maior relevância quanto mais independentes sejam os modelos construídos. Assim por exemplo, as ações do modelo dinâmico devem constituir um subconjunto dos métodos dos objetos. O mesmo acontece com as funções do modelo funcional em relação aos métodos, e os fluxos de dados com relação aos atributos e objetos.

A *completeza* diz relação com a determinação de se a modelagem está completa, i.e. poder saber se ainda restam componentes a serem incluídas na construção dos modelos.

Finalmente, a *não-ambigüidade* preocupa-se com a fato de existir uma única interpretação dos componentes utilizados nas modelagens.

3.1.2.4. O Critério de Comparação para o Procedimento da Análise

Este critério centra-se no procedimento que apresenta a técnica de análise, ou seja, quais são especificamente os passos que devem ser realizados para completar a tarefa de análise.

Este procedimento pode estabelecer ordens específicas para a construção dos modelos, o qual redunda na integração dos mesmos.

A tabela 3.1 resume a estrutura de todos os critérios de comparação descritos para o processo de análise.

3.2. Os Critérios de Comparação para a Especificação e Representação da Análise

Os critérios de esta categoria estão associados com a especificação e representação utilizada na análise. Da mesma maneira que os critérios de comparação do processo de análise, também foram divididos de acordo com os aspectos ou dimensões na construção dos modelos. Assim, estes critérios são categorizados em:

- aqueles associados às representações das dimensões da modelagem, e
- aqueles relativos à especificação como um todo.

3.2.1. Os Critérios de Comparação para a Representação das Dimensões

Analogamente aos critérios da modelagem, os critérios de comparação para a representação são divididos conforme as dimensões estrutural, dinâmica e funcional.

Os critérios para estas três dimensões são os mesmos, e são classificados segundo a seguinte estrutura:

- representação dos componentes e relações entre eles para cada dimensão,
- representação de restrições nos modelos de cada dimensão, e
- representação de níveis de abstração em cada dimensão.

1. Modelagem
    - 1.1. Dimensão Estrutural dos Objetos
        - 1.1.1. Identificação de Componentes
            - classes
            - atributos
            - métodos
            - relacionamentos
        - 1.1.2. Localização de Componentes
            - classes
            - atributos
            - métodos
        - 1.1.3. Particionamento da Complexidade Estrutural
    - 1.2. Dimensão Dinâmica do Comportamento
        - 1.2.1. Identificação de Componentes
            - estados
            - transições
            - eventos
            - ações (métodos da dimensão estrutural)
        - 1.2.2. Localização de Componente
            - ações
        - 1.2.3. Particionamento da Complexidade Dinâmica
    - 1.3. Dimensão Funcional dos Requisitos
        - 1.3.1. Identificação de Componentes
            - funções (métodos da dimensão estrutural)
            - fluxos de dados
        - 1.3.2. Particionamento da Complexidade Funcional
2. Integração das Dimensões
    - 2.1. Reusabilidade
    - 2.2. Construção de Modelos
    - 2.3. Verificações
        - consistência
        - completeza
        - não-ambigüidade
    - 2.4. Procedimento da Análise

Tabela 3.1 - Hierarquia de critérios de comparação para o processo de análise.

3.2.1.1. Os Critérios de Comparação para a Representação da Dimensão Estrutural dos Objetos

No caso desta dimensão, deve ser feita a *representação dos componentes* a seguir:

- *Classes*: relaciona-se com a notação utilizada para representar as classes.
- *Atributos/Métodos*: notação utilizada para atributos e/ou métodos nas classes.
- *Relacionamentos*: notações para os diferentes tipos de relacionamentos identificados.
- *Restrições*: tem a ver com possíveis restrições introduzidas na estrutura modelada dos objetos. Algumas restrições comuns são as cardinalidades das associações e os domínios dos atributos.
- *Níveis de Abstração*: deve responder à pergunta: Como representar o particionamento da complexidade estrutural de forma de facilitar a construção e a compreensão da especificação?

3.2.1.2. Os Critérios de Comparação para a Representação da Dimensão Dinâmica do Comportamento

A *representação dos componentes* nesta dimensão considera o seguinte:

- *Estados/Transições*: relaciona-se com a notação utilizada para representar os estados e/ou as transições entre estes estados.
- *Eventos/Ações*: notação utilizada para eventos e/ou ações nas classes.
- *Controle/Comunicação*: diz relação com a emissão/recepção de eventos (controle) e mensagens (comunicação) entre os objetos. A comunicação representa a troca de mensagens que especifica o comportamento dos objetos em tempo de execução, e por esta razão está mais vinculado com o POO.
- *Restrições*: tem a ver com possíveis restrições introduzidas no comportamento dinâmico dos objetos. Algumas restrições comuns são a inibição ou ativação de transições de acordo com certas condições.

- *Níveis de Abstração*: deve responder à pergunta: Como representar o particionamento da complexidade dinâmica ou *comportamental* dos objetos de forma de facilitar a construção e a compreensão da especificação?

3.2.1.3. Os Critérios de Comparação para a Representação da Dimensão Funcional dos Requisitos

A *representação dos componentes* nesta dimensão considera o seguinte:

- *Funções*: está relacionado com a notação utilizada para representar as funções ou processos no contexto desta dimensão.
- *Fluxos de Dados*: notação utilizada para os fluxos de dados de entrada e/ou de saída das funções.
- *Restrições*: tem a ver com possíveis restrições introduzidas na função de transformação dos objetos. Algumas restrições comuns são a concorrência ou exclusividade na recepção ou geração de fluxos de dados.
- *Níveis de Abstração*: deve responder à pergunta: Como representar o particionamento da complexidade funcional dos requisitos do sistema de objetos, de forma de facilitar a construção e compreensão da especificação?

3.2.2. Os Critérios de Comparação para a Especificação Global

Estes critérios tem a ver com características que possui a especificação orientada a objetos como um todo. Os critérios de comparação são: a quantidade de modelos, a complexidade aparente e a adaptabilidade ao domínio da aplicação.

O critério de comparação definido para a *quantidade de modelos* relaciona-se com o número final de modelos, *templates*, diagramas, documentos, etc. que configuram a especificação.

A *complexidade aparente* pretende avaliar a complexidade percebida desde o ponto de vista do usuário que participa da análise. Em outras palavras, a especificação é de fácil compreensão por parte do pessoal não especialista?

Finalmente, o critério de comparação para a *adaptabilidade ao domínio de aplicação* centra-se no grau de flexibilidade permitida pela técnica para adaptar, principalmente as notações ou representações, aos requisitos que podem demandar os diferentes domínios de aplicação. Assim por exemplo, uma notação própria para sistemas comerciais poderia não ser a mais adequada para representar um sistema de projetos de engenharia.

A tabela 3.2 sintetiza todos os critérios de comparação da categoria de especificação e representação da análise.

Isto completa a definição de critérios, o capítulo seguinte mostra a comparação de diversas técnicas de AOO em base a estes critérios de comparação.

1. Representação
    1.1. Dimensão Estrutural dos Objetos
        - classes
        - atributos/métodos
        - relacionamentos
        - restrições
        - níveis de abstração
    1.2. Dimensão Dinâmica do Comportamento
        - estados/transições
        - eventos/ações
        - controle/comunicação
        - restrições
        - níveis de abstração
    1.3. Dimensão Funcional dos Requisitos
        - funções
        - fluxos de dados
        - restrições
        - níveis de abstração
2. Especificação Global
    - quantidade de modelos
    - complexidade aparente
    - adaptabilidade ao domínio de aplicação

Tabela 3.2 - Hierarquia de critérios de comparação para a especificação e representação da análise.

4. COMPARAÇÃO DAS TÉCNICAS DE AOO

Este capítulo apresenta a comparação das técnicas descritas no capítulo 2, utilizando os critérios definidos no capítulo 3. A comparação será feita em base a tabelas, que sintetizam conjuntos de critérios logicamente agrupados.

Nas comparações apresentadas neste capítulo não é considerada a técnica representativa da categoria evolutiva orientada à dinâmica. A proposta de Kappel [KAP 91] está insuficentemente descrita na literatura (ver seção 2.2.2.3 do capítulo 2) como para permitir uma comparação equilibrada com as outras técnicas de AOO.

As tabelas da 4.1 à 4.3 mostram as comparações das técnicas segundo os critérios do processo de análise para a modelagem das dimensões: estrutural dos objetos, dinâmica do comportamento e funcional dos requisitos, respectivamente.

MODELAGEM DA DIMENSÃO ESTRUTURAL DOS OBJETOS	TÉCNICAS TEXTUAIS (PRESSMAN)	TÉCNICAS EVOLUTIVAS ORIENTADAS A DADOS (COAD & YOURDON)	TÉCNICAS EVOLUTIVAS ORIENTADAS ÀS FUNÇÕES (BAILIN)	TÉCNICAS EVOLUTIVAS ORIENTADAS À DINÂMICA (KAPPEL)	TÉCNICAS INTEGRACIONISTAS (RUMBAUGH ET AL.)	TÉCNICAS REVERSAS (NERSON)	TÉCNICAS COMPORTAMENTAIS (RUBIN & GOLDBERG)
IDENTIFICAÇÃO DE COMPONENTES							
classes	Sublinhar nomes e cláusulas nominais.	Procurar em estruturas, outros sistemas, dispositivos, eventos, papéis, procedimentos operacionais, lugares e unidades organizacionais.	Objetos nos nomes dos processos (ação + objeto) de um DFD preliminar.		Listar todas as classes possíveis, eliminar as redundantes, irrelevantes, vagas, atributos, operações, papéis e implementações.	Usuários, unidades organizacionais, representações de tipos abstratos de dados com estados internos e serviços a oferecer.	Escolher entre iniciadores e participantes de comportamentos do sistema.
atributos	Sublinhar adjetivos.	Propriedades: conceitos atômicos.	Implícito na modelagem ER documentado com DD.		Listar propriedades do objetos, eliminar objetos, qualificadores, nomes, identificadores, atributos de associações, e valores internos.	Informações das classes (*questões* no diagrama de classe).	Examinar os valores receptores de serviços.

métodos	Sublinhar verbos, frases verbais e predicados.	Estados do objetos e serviços requeridos.	Decomposição de entidades e/ou funções nos DFDE's.		Não identificados explicitamente.	Não identificados.	Examinar serviços fornecidos e/ou contratados.
relacionamentos	Não suportado.	Potenciais generalizações e especializações para cada classe. Montagem-partes, recipiente-conteúdos e conjunto-membros para agregação.	Implícito na modelagem ER.		Listar associações possíveis, eliminar irrelevantes, de implementação, ações, ternárias e derivadas.	Relacionamentos intra e inter *clusters*.	Serviços con-tratados nos cartão de mo-delagem de objeto, servi-ços e propriedades comuns (herança) ou fatorização de múltiplas responsabilidades.
LOCALIZAÇÃO DE COMPONENTES							
classes	Não suportado.	Guias gerais em estruturas Gen-Espec e Todo-Parte.	Implícito na modelagem ER.		Guias gerais em hierarquias de herança.	Não suportado.	Não suportado
atributos	Examinar o nome que o adjetivo acompanha.	Associar diretamente. Em estruturas Gen-Espec tentar colocar na classe superior.	Implícito na modelagem ER.		Implícito na identificação.	Implícito na identificação.	Propriedades lógicas dos *parties* (iniciadores ou participantes) requeridos por contratos ou serviços.
métodos	Examinar o nome que realiza a ação (verbo).	Localizar no objeto que possui estados (atributos).	Sempre pertencem à entidade decomposta.		Não são localizados explicitamente.	Não são identificados.	Serviços contratados e/ou fornecidos.
Particionamento da Complexidade Estrutural	Não suportado.	Conceito de assunto: classes no topo da hierarquias com relativa independência.	Conceito de domínio: sub-domínios per-tencentes ao domínio do problema.		Conceito de módulo: conjunto de clas-ses que cap-tura um sub-conjunto ló-gico do mo-delo.	Conceito de *clusters*: agrupamento de classes de acordo à funcionalidade do sistema, a um nível de abstração ou local geográfico.	Conceito de área de atividade central: áreas do sistema que requerem análise e construção de *scripts*

Abreviaturas usadas na tabela:

DD = Dicionário de Dados
DER = Diagrama Entidade-Relacionamento
DFD = Diagrama de Fluxo de Dados
DFDE = Diagrama de Fluxo de Dados de Entidades
ER = Entidade-Relacionamento
Gen-Espec = Generalização-Especialização

Tabela 4.1 - Comparação das técnicas segundo os critérios do processo de análise para a modelagem estrutural dos objetos.

MODELAGEM DA DIMENSÃO DINÂMICA DO COMPORTAMENTO	TÉCNICAS TEXTUAIS (PRESSMAN)	TÉCNICAS EVOLUTIVAS ORIENTADAS A DADOS (COAD & YOURDON)	TÉCNICAS EVOLUTIVAS ORIENTADAS ÀS FUNÇÕES (BAILIN)	TÉCNICAS EVOLUTIVAS ORIENTADAS À DINÂMICA (KAPPEL)	TÉCNICAS INTEGRACIONISTAS (RUMBAUGH ET AL.)	TÉCNICAS REVERSAS (NERSON)	TÉCNICAS COMPORTAMENTAIS (RUBIN & GOLDBERG)
IDENTIFICAÇÃO DE COMPONENTES							
estados	Não suportado.	Verificar valores potenciais dos atributos.	Não suportado.		Intervalo entre a ocorrência de dois eventos consecutivos em um cenário.	Não suportado.	Verificar expressões de pré e pós-condições nos *scripts* (funções lógicas sobre atributos e valores).
transições	Não suportado.	Examinar alterações nos valores dos atributos.	Não suportado.		Limite entre dois estados consecutivos em um cenário.	Não suportado.	Mudanças de estados para estados no ciclo de vida do objeto.
eventos	Não suportado.	Interpretar envios de mensagens.	Não suportado.		Procurar por sinais, entradas, decisões, interrupções, transições e ações.	Ações iniciadas por agentes externos que produzem entrada de fluxos de dados. Estímulos relacionados com o tempo.	Cada *script* corresponde a um evento.
ações	Não suportado.	Não suportado.	Não suportado.		Atividade contínua (consume tempo) associada a um estado ou ação (instantânea) associada a uma transição.	Não suportado.	Não identificadas explicitamente.
LOCALIZAÇÃO DE COMPONENTE							
ações	Não suportado.	Não suportado.	Não suportado.		Duração e associação da ação (quando?)	Não suportado.	Não suportado.
Particionamento da Complexidade Dinâmica	Não suportado.	Particionamento por Classes-&-Objetos.	Não suportado.		Aninhamento de diagramas de estado por generalização de estados ou eventos.	Cenários para interações típicas.	Aninhamento de *state-charts*.

Tabela 4.2 - Comparação das técnicas segundo os critérios do processo de análise para a modelagem dinâmica do comportamento.

MODELAGEM DA DIMENSÃO FUNCIONAL DOS REQUISITOS	TÉCNICAS TEXTUAIS (PRESSMAN)	TÉCNICAS EVOLUTIVAS ORIENTADAS A DADOS (COAD & YOURDON)	TÉCNICAS EVOLUTIVAS ORIENTADAS ÀS FUNÇÕES (BAILIN)	TÉCNICAS EVOLUTIVAS ORIENTADAS À DINÂMICA (KAPPEL)	TÉCNICAS INTEGRACIONISTAS (RUMBAUGH ET AL.)	TÉCNICAS REVERSAS (NERSON)	TÉCNICAS COMPORTAMENTAIS (RUBIN & GOLDBERG)
IDENTIFICAÇÃO DE COMPONENTES							
fluxos de dados	Não suportado.	Não suportado.	Implícito na construção de DFD's e DFDE's.		Implícito na construção de DFD's.	Não suportado.	Não suportado.
funções	Não suportado.	Não suportado.	Implícito na construção de DFD's e DFDE's.		Implícito na construção de DFD's.	Não suportado.	Não suportado.
Particionamento da Complexidade Funcional	Não suportado.	Não suportado.	Feita simultaneamente com a decomposição de entidades para subentidades e para funções.		Decomposição funcional *top-down* ou refinamento sucessivo nos DFD's.	Não suportado.	Não suportado.

Abreviaturas usadas na tabela:
DFD = Diagrama de Fluxo de Dados
DFDE = Diagrama de Fluxo de Dados de Entidades

Tabela 4.3 - Comparação das técnicas segundo os critérios do processo de análise para a modelagem funcional dos requisitos.

A tabela 4.4 mostra a comparação das técnicas segundo os critérios da integração das modelagens apresentadas nas tabelas anteriores.

INTEGRAÇÃO DAS DIMENSÕES	TÉCNICAS TEXTUAIS (PRESSMAN)	TÉCNICAS EVOLUTIVAS ORIENTADAS A DADOS (COAD & YOURDON)	TÉCNICAS EVOLUTIVAS ORIENTADAS ÀS FUNÇÕES (BAILIN)	TÉCNICAS EVOLUTIVAS ORIENTADAS À DINÂMICA (KAPPEL)	TÉCNICAS INTEGRACIONISTAS (RUMBAUGH ET AL.)	TÉCNICAS REVERSAS (NERSON)	TÉCNICAS COMPORTAMENTAIS (RUBIN & GOLDBERG)
Reusabilidade	Não suportado.	Verificar resultados em OOA's anteriores em domínios de problemas iguais ou semelhantes nos diversos passos.	Não suportado.		Não suportado.	Suportado na representação do Modelo Estático.	Não suportado.

Construção de Modelos	Não suportado.	Diagrama de Classes-&-Objetos e Diagrama de Serviços (que inclui o Diagrama de Estado do Objeto)	DFDE e DER (documentado com DD).		Modelo de Objetos, Modelo Dinâmico e Modelo Funcional.	Modelo Estático e Modelo Dinâmico.	Modelo de Objetos e Modela da Dinâmica do Sistema.
VERIFICAÇÕES							
consistência	Não suportado.	Serviços definidos a partir de estados e atributos do objeto.	Relação 1:1 entre entidades do DER e do DFDE.		Diagramas de Estado por classe. Ações do modelo dinâmico e funções do modelo funcional mais complexas representadas no modelo de objetos.	Não indicado explicitamente.	Não indicado explicitamente.
completeza	Não suportado.	Regras heurísticas em base a perguntas.	Não suportado.		Todos os eventos dos cenários para o modelo dinâmico. Todos os requisitos para o modelo funcional.	Não suportado.	Todos os *scripts* devem ser mapeados.
não-ambigüidade	Não suportado.	Implícito nos recursos de modelagem.	Implícito nos recursos de modelagem.		Implícito nos recursos de modelagem.	Implícito nas representações utilizadas.	Implícito nas representações utilizadas.
Procedimento da Análise	1) Descrever sistema com estratégia informal. 2) Sublinhar nomes como objetos. 3) Sublinhar adjetivos como atributos. 4) Sublinhar verbos como métodos. 5) Sublinhar advérbios como atributos dos métodos.	1) Definir objetos e classes. 2) Definir estruturas. 3) Definir assuntos. 4) Definir atributos. 5) Definir serviços.	1) Identificar entidades. 2) Distinguir entidades ativas de passivas. 3) Estabelecer fluxos de dados entre entidades ativas. 4) Decompor entidades (ou funções) em subentidades e/ou funções. 5) Verificar novas entidades. 6) Agrupar funções sob as novas entidades. 7) Definir domínios para as entidades.		1) Construir modelo de objetos. 2) Construir modelo dinâmico. 3) Construir modelo funcional. 4) Acrescentar operações nos modelos.	1) Identificar, nomear e *clustering* de classes. 2) Identificar eventos e definir protocolos de comunicação de objetos. 3) Definir classes e projetar arquitetura básica.	1) Definir o contexto da análise. 2) Entender o problema. 3) Definir os objetos. 4) Classificar os objetos e identificar os relacionamentos. 5) Modelar a dinâmica do sistema.

Abreviaturas usadas na tabela:

DD = Dicionário de Dados

DER = Diagrama Entidade-Relacionamento
DFDE = Diagrama de Fluxo de Dados de Entidades
OOA = Análise Orientada a Objetos

Tabela 4.4 - Comparação das técnicas segundo os critérios do processo de análise para a integração das dimensões da modelagem.

As tabelas da 4.5 à 4.7 mostram as comparações das técnicas segundo os critérios da especificação e representação da análise para as dimensões: estrutural dos objetos, dinâmica do comportamento e funcional dos requisitos, respectivamente.

REPRESENTAÇÃO DA DIMENSÃO ESTRUTURAL DOS OBJETOS	TÉCNICAS TEXTUAIS (PRESSMAN)	TÉCNICAS EVOLUTIVAS ORIENTADAS A DADOS (COAD & YOURDON)	TÉCNICAS EVOLUTIVAS ORIENTADAS ÀS FUNÇÕES (BAILIN)	TÉCNICAS EVOLUTIVAS ORIENTADAS À DINÂMICA (KAPPEL)	TÉCNICAS INTEGRACIONISTAS (RUMBAUGH ET AL.)	TÉCNICAS REVERSAS (NERSON)	TÉCNICAS COMPORTAMENTAIS (RUBIN & GOLDBERG)
classes	Coluna na Tabela de Objetos.	Caixa de cantos arredondados dividida em 3 seções: nome da classe na primeira seção no Diagrama de Classes-&-Objetos.	Entidades (notação comum) no DER e círculos ou fluxos de dados com nome entre colchetes nos DFDE's.		Caixa com 3 seções: nome da classe na primeira seção no Modelo de Objetos.	Elipses com o nome da classe dentro no Modelo Estático.	Caixas com o nome dentro nos Diagramas de Relacionamentos Organizacionais.
atributos/métodos	Coluna na Tabela de Objetos.	Nomes na segunda e terceira seção da classe, respectivamente no Diagrama de Classes-&-Objetos.	Atributos apenas documentados em um DD. Métodos como funções e subfunções (notação comum) nos DFDE's.		Nomes na segunda e terceira seção da classe, respectivamente no Modelo de Objetos.	Não representados.	Apenas representados nos Cartões de Modelagem de Objeto.
relacionamentos	Não suportado.	Arcos entre as classes: Gen-Espec, Todo-Parte e ocorrência nos Diagramas de Classes-&-Objetos.	Relacionamentos (notação comum) nos DER.		Arcos para associações específicas, com qualificação, generalização, agregação, ordenamento e associação ternária no Modelo de Objetos.	Setas entre classes ou *clusters* para: herança simples, relacionamentos cliente/fornecedor (associações e agregações) no Modelo Estático.	Setas para mostrar herança nos Diagramas de Relacionamentos Organizacionais.
restrições	Não suportado.	Cardinalidades nos relacionamentos: limite inferior e superior no Diagrama de Classes-&-Objetos.	Não suportado.		Notações para cardinalidade, restrições nos objetos e nas associações no Modelo de Objetos.	Símbolos: (*) classes diferidas, (+) para classes não diferidas e classes reusadas são sublinhadas no Modelo Estático.	Não suportado.

níveis de abstração	Não suportado.	Assuntos: borda em torno das classes relacionadas no Diagrama de Classes-&-Objetos.	Um DER para cada subdomínio do problema.		Módulos: ordenados por páginas para o Modelo de Objetos.	*Clusters*: caixas de cantos arredondados (com nome incluído) em torno das classes relacionadas no Modelo Estático.	Não suportado.

Abreviaturas usadas na tabela:

DD = Dicionário de Dados

DER = Diagrama Entidade-Relacionamento
DFDE = Diagrama de Fluxo de Dados de Entidades
Gen-Espec = Generalização-Especialização

Tabela 4.5 - Comparação das técnicas segundo os critérios da representação e especificação da análise para a dimensão estrutural dos objetos.

REPRESENTAÇÃO DA DIMENSÃO DINÂMICA DO COMPORTAMENTO	TÉCNICAS TEXTUAIS (PRESSMAN)	TÉCNICAS EVOLUTIVAS ORIENTADAS A DADOS (COAD & YOURDON)	TÉCNICAS EVOLUTIVAS ORIENTADAS ÀS FUNÇÕES (BAILIN)	TÉCNICAS EVOLUTIVAS ORIENTADAS À DINÂMICA (KAPPEL)	TÉCNICAS INTEGRACIONISTAS (RUMBAUGH ET AL.)	TÉCNICAS REVERSAS (NERSON)	TÉCNICAS COMPORTAMENTAIS (RUBIN & GOLDBERG)
estados/transições	Não suportado.	Estados como caixas e transições como setas no Diagrama de Estado do Objeto.	Não suportado.		Estados como caixas de cantos arredondados com o nome dentro e transições como arcos entre os estados nos Diagramas de Estado (notação dos *statecharts*) do Modelo Dinâmico.	Não suportado.	Estados e transições (notação dos *statecharts*) no Modelo da Dinâmica do Sistema.
eventos/ações	Não suportado.	Não suportado	Não suportado.		Eventos sobre as transições e ações nas transições ou nos estados nos Diagramas de Estado do Modelo Dinâmico.	Eventos como setas entre classes no Modelo Dinâmico. Ações não representadas.	Eventos com notação de *statechart* no Modelo da Dinâmica do Sistema. Ações não são representadas .
controle/comunicação	Não suportado.	Setas entre classes como conexões de mensagens no Diagrama de Classes-&-Objetos.	Não suportado.		Mesmo nome do evento em Diagramas de Estado de classes diferentes do Modelo Dinâmico.	Seqüências numeradas nas setas que representam os eventos no Modelo Dinâmico.	Setas nos Diagramas de Relacionamentos Contratuais.

restrições	Não suportado.	Não suportado.	Não suportado.		Eventos com atributos, estados inicial e final, e transições condicionadas nos Diagramas de Estado do Modelo Dinâmico.	Não suportadas.	Não suportadas.
níveis de abstração	Não suportado.	Um Diagrama de Estado por classe.	Não suportado.		Superestados em torno de subestados, agrupamento de transições nos Diagramas de Estado do Modelo Dinâmico.	Não suportadas.	Aninhamento de estados e fatorização de transições (notação de *statecharts*) no Modelo da Dinâmica do Sistema.

Tabela 4.6 - Comparação das técnicas segundo os critérios da representação e especificação da análise para a dimensão dinâmica do comportamento.

REPRESENTAÇÃO DA DIMENSÃO FUNCIONAL DOS REQUISITOS	TÉCNICAS TEXTUAIS (PRESSMAN)	TÉCNICAS EVOLUTIVAS ORIENTADAS A DADOS (COAD & YOURDON)	TÉCNICAS EVOLUTIVAS ORIENTADAS ÀS FUNÇÕES (BAILIN)	TÉCNICAS EVOLUTIVAS ORIENTADAS À DINÂMICA (KAPPEL)	TÉCNICAS INTEGRACIONISTAS (RUMBAUGH ET AL.)	TÉCNICAS REVERSAS (NERSON)	TÉCNICAS COMPORTAMENTAIS (RUBIN & GOLDBERG)
funções	Não suportadas.	Não suportadas.	Notação dos DFD's nos DFDE's.		Notação dos DFD's no Modelo Funcional.	Não suportadas.	Não suportadas.
fluxos de dados	Não suportadas.	Não suportadas.	Notação de DFD's nos DFDE's.		Notação dos DFD's no Modelo Funcional.	Não suportadas.	Não suportadas.
restrições	Não suportadas.	Não suportadas.	Não suportadas.		Fluxos de controle representados por setas de linhas pontilhadas no Modelo Funcional.	Não suportadas.	Não suportadas.
níveis de abstração	Não suportadas.	Não suportadas.	Árvore de decomposição funcional com DFDE's nivelados (como nos DFD's).		Árvore de decomposição funcional com diagramas nivelados no Modelo Funcional (como nos DFD's).	Não suportadas.	Não suportadas.

Abreviaturas usadas na tabela:

DFD = Diagrama de Fluxo de Dados
DFDE = Diagrama de Fluxo de Dados de Entidades

Tabela 4.7 - Comparação das técnicas segundo os critérios da representação e especificação da análise para a dimensão funcional dos requisitos.

A tabela 4.8 mostra as comparações das técnicas para os critérios da especificação global da analise.

ESPECIFICAÇÃO GLOBAL	TÉCNICAS TEXTUAIS (PRESSMAN)	TÉCNICAS EVOLUTIVAS ORIENTADAS A DADOS (COAD & YOURDON)	TÉCNICAS EVOLUTIVAS ORIENTADAS ÀS FUNÇÕES (BAILIN)	TÉCNICAS EVOLUTIVAS ORIENTADAS À DINÂMICA (KAPPEL)	TÉCNICAS INTEGRACIONISTAS (RUMBAUGH ET AL.)	TÉCNICAS REVERSAS (NERSON)	TÉCNICAS COMPORTAMENTAIS (RUBIN & GOLDBERG)
quantidade de modelos	1 tabela (Tabela de Objetos)	2 conjuntos de diagramas (Diagramas de Classes-&-Objetos e Diagramas de Serviços)	2 conjuntos de diagramas (DER's e DFDE's)		3 conjuntos de diagramas (Modelo de Objetos, Modelo Dinâmico e Modelo Funcional)	1 conjunto de tabelas (Diagramas de Classes), 1 conjunto de diagramas (Modelo Dinâmico), 1 diagrama (Modelo Estático)	1 conjunto de tabelas (*scripts*), 1 conjunto de glossários, 1 conjunto de diagramas (Modelo de Objetos), 1 conjunto de diagramas (Modelo da Dinâmica do Sistema)
complexidade aparente	Muito simples.	Simples.	Simples.		Complexo.	Medianamente complexo.	Medianamente complexo.
adaptabilidade ao domínio de aplicação	Restrito à definição de uma estratégia informal.	Modificações às notações são permitidas.	Flexibilidade inerente aos DER's e DFDE's.		Flexibilidade própria dos modelos usados. Algumas modelagens podem ser mínimas ou nulas dependendo do domínio.	Problemas com solução implementada na linguagem de programação *Eiffel*.	Modificações e extensões às representações e modelos são permitidas e ainda recomendadas.

Abreviaturas usadas na tabela:
DER = Diagrama Entidade-Relacionamento
DFDE = Diagrama de Fluxo de Dados de Entidades

Tabela 4.8 - Comparação das técnicas segundo os critérios da especificação global da análise.

5. PONTOS FRACOS E PONTOS FORTES NAS CATEGORIAS E TÉCNICAS DE AOO

Uma vez concluída a comparação das técnicas que representam as categorias de AOO, é possível indicar alguns características que merecem atenção, especialmente porque ainda não estão bem desenvolvidas nas propostas apresentadas.

5.1. AS CRÍTICAS ÀS CATEGORIAS DE TÉCNICAS DE AOO

Em primeiro lugar, é recomendável avaliar a classificação proposta, que é mais detalhada que as referenciadas (ver capítulo 2, seção 2.1). Assim também é possível avaliar as técnicas que representam cada categoria, e facilitar a identificação dos pontos fortes e fracos nas propostas atuais na literatura. As categorias propostas foram: textuais, evolutivas orientadas a dados, funções e dinâmica, integracionistas, reversas e comportamentais.

As técnicas textuais estão definitivamente obsoletas como técnicas de AOO por si mesmas, mas podem auxiliar nos passos iniciais das técnicas mais modernas.

As técnicas evolutivas orientadas a dados parecem ser as mais desenvolvidas, graças à bagagem da modelagem semântica ou modelagem de informação que utilizam. Os modelos são mais estáveis, e já possuem um considerável consenso nos procedimentos, e uma relativa uniformidade nas notações. Estas técnicas são inclusive recomendadas por autores de prestígio da área da análise e projeto estruturado, como Yourdon [COA 92] e [YOU 90a], e Constantine [DEC 90]. Porém, sua aplicabilidade exige a construção de modelos nos outros aspectos da modelagem. Por enquanto, parecem ser mais úteis para domínios de problemas que requerem uma modelagem de bancos de dados.

As técnicas evolutivas orientadas às funções encontram-se em algum ponto da transição entre as técnicas atuais (análise, projeto e programação estruturados) e a tecnologia de orientação a objetos. Hoje parecem ser mais úteis em domínios de problemas que apresentam muito cálculo (numerosas funções de transformação e conversão), como por exemplo aplicações matemáticas ou científicas de tempo real.

As técnicas evolutivas orientadas à dinâmica também parecem ter um futuro promissor. As aplicações atuais põem mais ênfase nas interfaces gráficas ao usuário (*GUI*) e os domínios são cada vez mais voltados à dinâmica da interação com entidades externas ao sistema de software. Porém, as propostas revistas ainda precisam de maior maturidade e simplicidade.

As técnicas integracionistas visam uma maior aplicação em domínios diferentes. Dependendo do tipo de problema, a ênfase em determinados aspectos da modelagem pode ser maior ou menor. A maior dificuldade aparece na hora de integrar visões que podem ser muito independentes. Isto dificulta a transição ao POO.

As técnicas reversas buscam uma maior eficiência no desenvolvimento em linguagens específicas de programação orientadas a objetos, sacrificando a generalidade que as aplicações requerem. Podem constituir alternativas válidas quando a decisão sobre a implementação já está tomada no momento de iniciar o desenvolvimento. Porém parece importante um maior maturidade e/ou integração com técnicas de outras categorias para melhorar a efetividade das especificações.

Finalmente, as técnicas comportamentais aportam um conceito interessante: a definição de responsabilidades dos objetos ao interior do sistema. A estratégia de identificar objetos, e por ende responsabilidades, a partir do comportamento esperado ou desejado do sistema é digno de consideração nas fases iniciais da AOO, porém parece que falta uma maior ênfase na modelagem nas técnicas pertencentes a esta categoria.

5.2. O PONTO FORTE DAS TÉCNICAS DE AOO

Sem dúvida nenhuma o ponto forte da maioria das técnicas está na dimensão estrutural dos objetos. Como foi indicado na seção anterior, a principal causa disto é que a maioria das propostas utilizam técnicas de modelagem semântica baseadas em modelos estendidos de entidades e relacionamentos. Estas técnicas possuem um desenvolvimento de mais de 15 anos, a partir do trabalho original de Chen em 1976 [CHE 76].

A curva de crescimento desta tecnologia (modelagem estrutural) parece estar na fase de estabilidade, portanto já possui a maturidade suficiente para ser utilizada com relativa facilidade.

Neste sentido, a identificação e especificação de objetos, classes, métodos, atributos e relacionamentos de herança, agregação e outros é o principal ponto forte das técnicas de AOO, com todas as dificuldades inerentes a este tipo de modelagem.

5.3. Os Pontos Fracos das Técnicas de AOO

Como em qualquer tecnologia emergente, os pontos fracos que apresentam são muito mais numerosos do que os pontos fortes. A orientação a objetos, e mais especificamente a AOO, não é a exceção.

Para ordenar as observações neste considerando, os pontos fracos serão divididos em pontos fracos a nível *macro* e pontos fracos a nível *micro*.

5.3.1. Os *Macro* Pontos Fracos das Técnicas de AOO

Estes pontos fracos dizem respeito a modificações, extensões ou inclusões a um nível global das técnicas, sem considerar detalhes específicos de procedimentos ou aspectos de modelagem.

Como *macro* pontos fracos podem ser indicados os seguintes:

- *Critérios consistentes de particionamento da complexidade*: Deve existir critérios objetivos e práticos que permitem agregar classes ou particionar sistemas. As primeiras questões que surgem são: O particionamento inicial deve ser na perspectiva da organização, da qual o sistema de informação faz parte (sistemas e subsistemas) ou na perspectiva do software orientado a objetos (classes e objetos)? São incompatíveis estas visões? Neste sentido também é válido questionar se o particionamento deve ser realizado *antes* ou *após* a identificação das classes, ou em outras palavras se o sistema é particionado ou as classes são agregadas.

Considerando esta última alternativa, a agregação das classes, existem na literatura diversas propostas (algumas já foram mencionadas na comparação do capítulo anterior), entre as quais podem ser indicadas: os *frameworks* [NIE 92] e [WIR 90], para o reuso de componentes; os assuntos [COA 92], que agrupam classes com relativa independência; os padrões (*patterns*) [COA 92a], que definem critérios para agrupar classes; os módulos [RUM 91], para uma especificação modularizada; os subsistemas [SHL 92] e [WIR 92]; os subdomínios [BAI 89]; os *clusters* [NER 92] e [MEY 88], conceito da linguagem *Eiffel*; os domínios [SHL 92]; e os *ensembles* [WIR 90] e [FIC 92], conceito de De Champeaux que agrupa classes em torno de estruturas de agregação e composição de maneira análoga aos objetos, podendo apresentar atributos, métodos, estados e transições, relacionamentos e classificação, porém com a diferença fundamental de que os *ensembles* podem apresentar paralelismo interno enquanto que os objetos não[8].

A única proposta para a alternativa de particionar antes da identificação de classes e objetos são as áreas de interesse [WIR 90], da técnica *OORASS* (descrita sucintamente nas técnicas reversas), lamentavelmente não está suficientemente documentada na literatura.

A solução para este ponto fraco deve permitir uma modelagem considerando níveis consistentes de abstração nos diferentes aspectos da especificação orientada a objetos.

- *Reusabilidade na especificação*: Esta questão ainda não recebeu a devida atenção na literatura. As propostas de AOO's mais *ousadas* apenas recomendam estudar outras especificações em domínios iguais ou semelhantes [COA 92], porém não fornecem nenhum mecanismo concreto para identificar e avaliar as classes potenciais para reuso.

  Este ponto fraco parece paradoxal se é considerado o fato que um dos fundamentos da introdução da tecnologia de orientação a objetos está no

[8] Este critério pode parecer mais de implementação, porém parece mais objetivo que outros mecanismos propostos na literatura.

maior grau de reusabilidade que promete. Nas fases de projeto, e particularmente na implementação, este conceito está mais desenvolvido do que na AOO.

O reuso na AOO, também denominado reuso *harvesting*, pode tomar duas formas: reuso de especificações prévias de análise e projeto orientados a objetos ou abstrações de componentes de programas já implementados (engenharia reversa) [FIC 92].

O reuso a nível de especificação parece difícil por dois motivos: 1) a AOO trata com domínios de problemas e não componentes de software, portanto o máximo reuso pode ocorrer no interior de domínios específicos de aplicação; e 2) as classes identificadas deveriam ser o mais genéricas possíveis para serem armazenadas em bibliotecas, isto pode gerar conflitos entre a especificidade que a aplicação demanda e genericidade que o potencial reuso requer.

- *Modelagem funcional v/s modelagem "end-to-end"*: A controvérsia já discutida no capítulo 1, seção 1.5.3, não está resolvida. A complexidade das aplicações exige modelagens funcionais para especificar os requisitos, mas as técnicas consideram que isto pode influir negativamente na *orientabilidade* do sistemas: funcional ou objetos?

  A questão chave parece ser o critério de particionamento no aspecto funcional, se for por decomposição funcional (estratégia *top-down* ou por refinamento sucessivo) este é claramente contrário à orientação a objetos. Se as funções forem subordinadas aos objetos, em cujo caso tornar-se-iam métodos, não há problemas de encapsulamento, e a especificação corresponderia a uma modelagem de processos *end-to-end* [FIC 92] e [BAI 89] ou modelagem funcional encapsulada.

- *Validação do usuário*: A maioria das técnicas não considera explicitamente a participação do usuário na modelagem do domínio do problema, ou na definição dos requisitos para a solução. Apenas Coad & Yourdon [COA 92] dão maior relevância a esta questão.

O usuário pode participar na construção e validação dos modelos da AOO, ou através do uso de protótipos da aplicação.

- *Estimação ou dimensionamento de sistemas*: A estimação do tamanho dos sistemas, considerando alguma métrica, é uma atividade fundamental nos projetos de engenharia de software, mas que parece não ser ainda considerada nas técnicas de AOO e POO.

  As propostas de métodos de estimação existentes [LAR 90] e [CHI 91], tratam este aspecto sem maior vinculação à modelagem e especificação de sistemas orientados a objetos. Uma adequada estimação do tamanho da solução a propor, considerando as classes identificadas no domínio do problema e a reusabilidade a nível de análise e projeto orientados a objetos, permite projetar o desenvolvimento nas seguintes fases.

5.3.2. Os *Micro* Pontos Fracos das Técnicas de AOO

Estes pontos fracos dizem relação com detalhes e considerações específicas que melhorariam as atuais técnicas de AOO.

Como *micro* pontos fracos podem ser indicados os seguintes:

- *Análise de domínio*: Já foi mencionada a vinculação que a AOO possui com o conceito relativamente novo da *análise de domínio* [ARA 89]. As perguntas chaves aqui são: É válido estudar o domínio do problema em forma independente do problema? E a reusabilidade no domínio como pode ser atingida senão? Das técnicas vistas, somente Coad & Yourdon [COA 92] sugerem tal consideração.

- *Interação dinâmica de classes*: As diversas técnicas não são, nem de longe, uniformes a este respeito. Algumas identificam as interações no modelo estrutural [COA 92], enquanto outras no modelo dinâmico [RUM 91], e outras até propõem um modelo separado para tal efeito [CLY 92] e [EMB 92]. A coordenação do comportamento das classes é fundamental para uma especificação no aspecto dinâmico.

- *Integração de modelos estáticos e dinâmicos*: Na maioria das propostas sempre existe dificuldades para integrar ou referenciar os aspectos estáticos e dinâmicos dos sistemas de objetos. Este ponto fraco é uma herança que antecede a orientação a objetos e ainda não está completamente resolvido na literatura.

- *Conhecimento na especificação*: Poderia ser considerado um outro aspecto na modelagem e especificação de sistemas orientados a objetos? A modelagem de conhecimento é um área fértil na inteligência artificial. É possível considerar por exemplo, a inclusão de regras nos objetos para que possam agir *inteligentemente* [PRI 91]. Os domínios de aplicação para estes sistemas seriam muito específicos.

- *Formalização da especificação*: Pode ser preciso definir formalmente a sintaxe e a semântica dos modelos de software orientados a objetos para garantir uma compreensão e comunicação precisas. Porém, a flexibilidade da informalidade também é desejável. Todas as propostas vistas são informais, com exceção da técnica *OSA* [CLY 92] que fornece uma definição informal e formal dos componentes dos modelos.

- *Métodos para avaliação da qualidade dos modelos*: Quanto maiores sejam os sistemas a serem desenvolvidos com a tecnologia de orientação a objetos, mais necessário será contar com técnicas e métodos que permitam avaliar, tanto em termos quantitativos como não quantitativos, a qualidade dos modelos construídos. Atualmente, no existe qualquer referência a esta questão nas propostas da literatura.

Outras deficiências mais específicas, considerando várias técnicas de AOO, podem ser consultadas em [ASK 92].

6. CONCLUSÕES E COMENTÁRIOS

Como conclusão podem ser examinados os objetivos iniciais para verificar se foram atingidos:

- Foi realizado um estudos das técnicas, categorias, enfoques e classificações na área da AOO.

- Foram estabelecidos um conjunto de critérios para comparar efetivamente as técnicas de AOO.

- Foi apresentada uma comparação detalhada das técnicas representativas de cada categoria de AOO utilizando para tal fim, os critérios previamente definidos.

- Foram indicadas os pontos fortes e os pontos fracos que apresentam as técnicas propostas como conjunto para a fase de AOO.

Assim os objetivos enunciados foram plenamente satisfeitos no desenvolvimento deste trabalho.

Em termos gerais, as técnicas revistas apresentam uma relativa falta de maturidade, questão por demais natural, dado o fato que o conceito de AOO é recente. Por isto, ainda não existe uma técnica já padronizada e de ampla aceitação que possa vir a substituir as metodologias convencionais, como a análise estruturada ou a engenharia da informação. Nem sequer é possível afirmar qual será a categoria, enfoque ou classificação que dominará no futuro.

Considerando o estado da arte em AOO, as categorias das técnicas evolutivas orientadas a dados e à dinâmica, nesta ordem, parecem ser as de futuro mais promissor. Devido, fundamentalmente, aos conceitos de modelagem em que se sustentam.

Para a fase de transição entre as metodologias convencionais e a tecnologia de orientação a objetos, as técnicas integracionistas e as técnicas evolutivas orientadas às funções, nesta ordem, parecem ser as mais razoáveis. Considerando o fato

significativo da resistência da indústria para adotar novas tecnologias. Isto agrava-se ao considerar que muitas empresas recentemente incorporaram técnicas estruturadas nos seus desenvolvimento de sistemas.

A AOO promete oferecer todo o seu potencial, com o máximo aproveitamento, se as fases seguintes também pertencerem ao paradigma da orientação a objetos. Em caso contrário, os mapeamentos e traduções entre as sucessivas fases, como por exemplo mapeamentos de modelos de objetos para modelos funcionais ou vice-versa, tornariam o desenvolvimento mais complexo do que o tradicional (ver figura 1.1 do capítulo 1).

Um fator importantíssimo para conseguir a aplicabilidade das técnicas de AOO, considerando aos fatos indicados acima, é a disponibilidade de ambientes CASE (*Computer-Aided Software Engineering*) que suportem todo o desenvolvimento orientado a objetos. Nestes ambientes, devem existir ferramentas que permitam a prototipação e verificação automática de consistência, não-ambigüidade e completeza dos modelos, além de fornecer editores diagramáticos e de texto para a elaboração das especificações, e repositórios e bibliotecas centralizados.

Como áreas para futuras pesquisas na AOO que venham a contribuir no fortalecimento dos pontos fracos, podem ser citadas as seguintes: definição de critérios de particionamento da complexidade estrutural (em consistência para os aspectos dinâmico e funcional), incorporação da reusabilidade a nível de especificação (análise de domínio), a modelagem funcional encapsulada (modelagem *end-to-end*), a integração de modelos estáticos e dinâmicos, a formalização dos modelos, e a definição de técnicas para avaliar a qualidade dos modelos construídos.

Uma técnica *ideal* de AOO deveria permitir uma modelagem e especificação precisa multidimensional (ver capítulo 1, figura 1.4) dos sistemas orientados a objetos, com distintos níveis de abstração em consistência com todos os aspectos dos modelos. A reusabilidade permitiria minimizar o esforço e especificar por extensão.

Finalmente, podem ser consultadas as referências [CHE 90], [DEC 92a], [FIC 92] e [MON 92] como um complemento para o estudo comparativo realizado neste trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

[ABB 83] ABBOTT, Russell. Program Design by Informal English Descriptions. **Communications of the ACM**, New York, v.26, n.11, p.882-894, Nov. 1983.

[ACK 91] ACKROYD, M.; DAUM, D. Graphical Notation for Object-Oriented Design and Programming. **Journal of Object-Oriented Programming**, New York, v.3, n.5, p.18-28, Jan. 1991.

[ALA 88] ALABISO, B. Transformation of Data Flow Analysis Models to Object-Oriented Design. **ACM SIGPLAN Notices**, New York, v.23, n.11, p.335-353, Nov. 198. Trabalho apresentado na Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA '88, 3., 1988, San Diego.

[ARA 89] ARANGO, Guillermo. Domain Analysis: From Art Form to Engineering Discipline. **ACM SIGSOFT Software Engineering Notes**, New York, v.14, n.3, p.152-159, May. 1989. Trabalho apresentado no International Workshop on Software Specification and Design, 5., 1989, Pittsburgh.

[ASK 92] ASKIT, Mehmet; BERGMANS, Lodewijk. Obstacles in Object-Oriented Software Development. **ACM SIGPLAN Notices**, New York, v.27, n.10, p.341-358, Oct. 1992. Trabalho apresentado na Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA '92, 7., 1992, Vancouver.

[BAI 89] BAILIN, Sidney. An Object-Oriented Requirements Specification Method. **Communications of the ACM**, New York, v.32, n.5, p.608-623, May. 1989.

[BEC 89] BECK, Kent; CUNNINGHAM, Ward. A Laboratory for Teaching Object-Oriented Thinking. **ACM SIGPLAN Notices**, New York, v.24, n.10, p.1-6, Oct. 1989. Trabalho apresentado na Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA '89, 4., 1989, New Orleans.

[BEA 90] BEAR, Stephen; ALLEN, Phillip; COLEMAN, Derek; HAYES, Fiona. Graphical Specification of Object-Oriented Systems. **ACM SIGPLAN Notices**, New York, v.25, n.10, p.28-37, Oct. 1990. Trabalho apresentado na Annual Conference of Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA '90, 5., 1990, Ottawa.

[BOO 83] BOOCH, Grady. Object-Oriented Design. In: FREEMAN, Peter; WASSERMAN, Anthony (eds.) **Tutorial on Software Design Techniques**, Los Alamitos: IEEE Computer Society Press, 1983, p.420-436.

[BOO 86] BOOCH, Grady. Object-Oriented Development. **IEEE Transactions on Software Engineering**, New York, v.12, n.2, p.211-221, Feb. 1986.

[BOO 91] BOOCH, Grady. **Object Oriented Design with Applications**. Redwood City: Benjamin/Cummings, 1991. 580p.

[BRU 92] BRUEGGE, Bernd; BLYTHE, Jim; JACKSON, Jeffrey; SHUFELT, Jeff. Object-Oriented System Modeling with OMT. **ACM SIGPLAN Notices**, New York, v.27, n.10, p.359-376, Oct. 1992. Trabalho apresentado na Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA '92, 7., 1992, Vancouver.

[BUL 89] BULMAN, David. An Object-Based Development Model. **Computer Language**, v.6, n.8, p.49-59, Aug. 1989.

[CAP 92] CAPRETZ, L.; LEE, P. A Classification of Object-Oriented Development Methodologies. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 6., 1992, Gramado. **Anais...** Gramado: II-UFRGS, 1992. 387p. p.129-141.

[CHE 76] CHEN, Peter. The Entity-Relationship Model: Toward a Unified View of Data. **ACM Transactions on Database Systems**, New York, v.1, n.1, p.9-36, Mar. 1976.

[CHE 90] CHEN, Jen-Yen; WANG, Jih-Jun. Comparing Object-Oriented Design Methods Experimentally. In: INTERNATIONAL CONFERENCE OF TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES & SYSTEMS - TOOLS 3, 3., 1990, Sidney. **Proceedings...** Sidney: TOOLS Pacific, 1990. p. 207-219.

[CHI 91] CHIDAMBER, Shyam; KEMERER, Chris. Towards a Metric Suite for Object-oriented Design. **ACM SIGPLAN Notices**, New York, v.26, n.11, p.197-211, Nov. 1991. Trabalho apresentado na Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA '91, 6., 1991, Phoenix.

[CLY 92] CLYDE, Stephen; EMBLEY, David; WOODFIELD, Scott. Tunable Formalism in Object-Oriented Systems Analysis: Meeting the Needs of Both Theoreticians and Practitioners. **ACM SIGPLAN Notices**, New York, v.27, n.10, p.452-465, Oct. 1992. Trabalho apresentado na Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA '92, 7., 1992, Vancouver.

[COA 92] COAD, Peter; YOURDON, Edward. **Análise Baseada em Objetos**. Rio de Janeiro: Campus, 1992. 225p.

[COA 92a] COAD, Peter. Object-Oriented Patterns. **Communications of the ACM**, New York, v.35, n.9, p.152-159, Sep. 1992.

[CUN 86] CUNNINGHAM, Ward; BECK, Kent. A Diagram for Object-Oriented Programs. **ACM SIGPLAN Notices**, New York, v.21, n.11, p.361-367, Sep. 1986. Trabalho apresentado na Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA '86, 1., 1986, Portland.

[DAV 88] DAVIS, Alan. A Comparison of Techniques for the Specification of External System Behavior. **Communications of the ACM**, New York, v.31, n.9, p.1098-1115, Sep. 1988.

[DEC 90] DE CHAMPEAUX, Dennis; CONSTANTINE, Larry; JACOBSON, Ivar; MELLOR, Stephen; WARD, Paul; YOURDON, Edward. Structured Analysis and Object-Oriented Analysis. **ACM SIGPLAN Notices**, New York, v.25, n.10, p.135-139, Oct. 1990. Painel acontecido na Annual Conference of Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA '90, 5., 1990, Ottawa.

[DEC 92] DE CHAMPEAUX, Dennis; LEA, Doug; FAURE, Penelope. The Process of Object-Oriented Design. **ACM SIGPLAN Notices**, New York, v.27, n.10, p.45-62, Oct. 1992. Trabalho apresentado na Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA '92, 7., 1992, Vancouver.

[DEC 92a] DE CHAMPEAUX, Dennis; FAURE, P. A Comparative Study of Object-Oriented Analysis Methods. **Journal of Object-Oriented Programming**, New York, v.5, n.1, p.21-33, 1992.

[DEM 78] DE MARCO, Tom. **Structured Analysis and System Specification**. New York: Yourdon Press, 1978.

[EMB 92] EMBLEY, David; KURTZ, Barry; WOODFIELD, Scott. **Object-Oriented System Analysis: A Model-Driven Approach**. Englewood Cliffs: Prentice-Hall, 1992. 302p.

[FIC 92] FICHMAN, Robert; KEMERER, Chris. Object-Oriented and Conventional Analysis and Design Methodologies: Comparison and Critique. **IEEE Computer**, Los Alamitos, v.25, n.10, p.22-39, Oct. 1992.

[FLA 81] FLAVIN, Matt. **Fundamental Concepts of Information Modeling**. New York: Yourdon Press, 1981. 128p.

[FRE 90] FREITAS, Maria; MOREIRA, Ana; GUERREIRO, Pedro. Object-Oriented Requirements Analysis in an Ada Project. **Ada Letters**, v.10, n.6, p.97-109, Jul./Aug. 1990.

[GAN 82] GANE, Chris; SARSON, Trish. **Structured System Analysis**. McDonell Douglas, 1982.

[GIB 90] GIBSON, Elizabeth. Objects - Born and Bred. **Byte**, Peterborough, v.15, n.10, p.245-254, Oct. 1990.

[GIR 90] GIRARDI, Maria; PRICE, Roberto. O Paradigma de Desenvolvimento por Objetos. **Revista de Informática Teórica e Aplicada**, Porto Alegre, v.1, n.2, p.69-98, May. 1990.

[GOS 90] GOSSAIN, Sanjiv; ANDERSON, Bruce. An Iterative-Design Model for Reusable Object-Oriented Software. **ACM SIGPLAN Notices**, New York, v.25, n.10, p.12-27, Oct. 1990. Trabalho apresentado na Annual Conference of Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA '90, 5., 1990, Ottawa.

[HAR 87] HAREL, David. Statecharts: A Visual Formalism for Complex Systems. **Science of Computer Programming**, Amsterdam, v.8, n.3, Jun. 1987.

[HAY 91] HAYES, Fiona; COLEMAN, Derek. Coherent Models for Object-Oriented Analysis. **ACM SIGPLAN Notices**, New York, v.26, n.11, p.171-183, Nov. 1991. Trabalho apresentado na Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA '91, 6., 1991, Phoenix.

[HEN 90] HENDERSON-SELLERS, Brian. Three Methodological Frameworks for Object-Oriented Systems Development. In: INTERNATIONAL CONFERENCE OF TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES & SYSTEMS - TOOLS 3, 3., 1990, Sidney. **Proceedings...** Sidney: TOOLS Pacific, 1990. p. 118-131.

[HEN 90a] HENDERSON-SELLERS, Brian; EDWARDS, Julian. The Object-Oriented Systems Life Cycle. **Communications of the ACM**, New York, v.33, n.9, p.142-159, Sep. 1990.

[HEU 90] HEUSER, Carlos. **Modelagem Conceitual de Sistemas: Redes de Petri**. Kapelusz: Buenos Aires, 1990.

[ISC 91] ISCOE, Neil; WILLIAMS, Gerald; ARANGO, Guillermo. Domain Modeling for Software Engineering. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 13., 1991. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 1991.

[JAC 83] JACKSON, Michael. **System Development**. London: Prentice-Hall, 1983.

[JAC 87] JACOBSON, Ivar. Object-Oriented Development in an Industrial Environment. **ACM SIGPLAN Notices**, New York, v.22, n.12, p.183-191, Dec. 1987. Trabalho apresentado na Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA '87, 2., 1987, Orlando.

[JAC 92] JACOBSON, Ivar. **Object-Oriented Software Engineering**. Reading: Addison-Wesley, 1992.

[JAL 89] JALOTE, Pankaj. Functional Refinement and Nested Objects for Object-Oriented Design. **IEEE Transactions on Software Engineering**, New York, v.15, n.3, p.264-270, Mar. 1989.

[KAP 91] KAPPEL, Gerti. Using an Object-Oriented Diagram Technique for the Design of Information Systems. In: SOL, H.; VAN HEE, K. (eds.) **Dynamic Modelling of Information Systems**, Amsterdam: North-Holland, 1991, p.121-164.

[KOR 90] KORSON, Timothy; McGREGOR, John. Understanding Object-Oriented: A Unifying Paradigm. **Communications of the ACM**, New York, v.33, n.9, p.40-60, Sep. 1990.

[KUR 89] KURTZ, Barry; HO, Donna; WALL, Teresa. An Object-Oriented Methodology for Systems Analysis and Specification. **Hewlett-Packard Journal**, Palo Alto, v.40, n.2, p.86-90, Apr. 1989.

[LAD 88] LADDEN, Richard. A Survey of Issues to be Considered in the Development of an Object-Oriented Development Methodology for Ada. **ACM SIGSOFT Software Engineering Notes**, New York, v.13, n.3, p.24-30, Jul. 1988.

[LAR 90] LARANJEIRA, Luiz. Software Size Estimation of Object-Oriented Systems. **IEEE Transactions on Software Engineering**, New York, v.16, n.5, p.510-522, May. 1990.

[LEE 91] LEE, S.; CARVER, D. Object-Oriented Analysis and Specification: A Knowledge Base Approach. **Journal of Object-Oriented Programming**, New York, p.35-43, Jan. 1991.

[LOO 87] LOOMIS, M.; SHAH, A.; RUMBAUGH, J. An Object Modeling Technique for Conceptual Design. In: EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING - ECOOP '87, 1987, Paris. **Proceedings...** Publicado em Lecture Notes in Computer Science, Berlin: Springer-Verlag, v.276, 1987. p.192-202.

[MAN 89] MANFREDI, F.; ORLANDI, G.; TORTORICI, P. An Object-Oriented Approach to the System Analysis. In: EUROPEAN SOFTWARE ENGINEERING CONFERENCE - ESEC '89, 2., 1989. **Proceedings...** Publicado em Lecture Notes in Computer Science, Berlin: Springer-Verlag, v.387, 1989. p.395-410.

[MAR 90] MARTIN, James. **Information Engineering: Books I, II & III**. Englewood Cliffs: Prentice-Hall, 1990.

[MAT 88] MATTOSO, Adriana; BLUM, Hélcio. Proposta de Desenvolvimento de Software com Orientação a Objetos. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 2., 1988, Canela. **Anais...** Canela, 1988. p.7-16.

[MCM 91] McMENAMIN, Stephen; PALMER, John. **Análise Essencial de Sistemas**. São Paulo: McGraw-Hill, 1991. 567p.

[MEY 88] MEYER, Bertrand. **Object-Oriented Software Construction**. Hertfordshire: Prentice-Hall, 1988. 534p.

[MON 92] MONARCHI, David; PUHR, Gretchen. A Research Typology for Object-Oriented Analysis and Design. **Communications of the ACM**, New York, v.35 n.9, p.35-47, Sep. 1992.

[NAV 89] NAVATHE, Shamkant; PILLALAMARRI, Mohan. OOER: Toward Making the E-R Approach Object-Oriented. In: BATINI, C. (ed.) **Entity-Relationship Approach**, Amsterdam: North-Holland, 1989, p.185-206.

[NER 91] NERSON, Jean-Marc. Extending Eiffel Toward Object-Oriented Analysis and Design. In: INTERNATIONAL CONFERENCE OF TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES & SYSTEMS - TOOLS 5, 5., 1990, Santa Barbara. **Proceedings...** Englewood Cliffs: Prentice Hall, 1991. p. 377-392.

[NER 92] NERSON, Jean-Marc. Applying Object-Oriented Analysis and Design. **Communications of the ACM**, New York, v.35, n.9, p.63-74, Sep. 1992.

[NIE 92] NIERSTRASZ, Oscar; GIBBS, Simon; TSICHRITZIS, Dennis. Component-Oriented Software Development. **Communications of the ACM**, New York, v.35, n.9, p.160-165, Sep. 1992

[PAG 90] PAGE-JONES, Meilir; CONSTANTINE, Larry; WEISS, Steven. Modeling Object-Oriented Systems: The Uniform Object Notation. **Computer Language**, v.7, n.10, p.69-87, Oct. 1990.

[PRE 87] PRESSMAN, Roger. **Software Engineering: A Practitioner's Approach**. 2nd ed., New York: McGraw-Hill, 1987.

[PRI 91] PRICE, Roberto. Engenharia de Software e Orientação a Objetos. In: WORKSHOP EM PROGRAMAÇÃO CONCORRENTE, SISTEMAS DISTRIBUÍDOS E ENGENHARIA DE SOFTWARE, 1991, São Carlos. **Anais...** São Carlos: ICMSC-USP, 1991. 131p. p.76-92.

[RIN 91] RINE, David. A Proposed Standard Set of Principles for Object-Oriented Development. **ACM SIGSOFT Software Engineering Notes**, New York, v.16, n.1, p.43-49, Jan. 1991.

[RUB 92] RUBIN, Kenneth; GOLDBERG, Adele. Object Behavior Analysis. **Communications of the ACM**, New York, v.35, n.9, p.48-62, Sep. 1992.

[RUM 91] RUMBAUGH, James; BLAHA, Michael; PREMERLANI, William; EDDY, Frederick; LORENSEN, William. **Object-Oriented Modeling and Design**. Englewood Cliffs: Prentice-Hall, 1991. 500p.

[SAK 88] SAKKINEN, M. Comments on "the Law of Demeter" and C++. **ACM SIGPLAN Notices**, New York, v.23, n.12, p.38-44, Dec. 1988.

[SCH 92] SCHASCHINGER, Harald. ESA - An Expert Supported OOA Method and Tool. **ACM SIGSOFT Software Engineering Notes**, New York, v.17, n.2, p.50-56, Apr. 1992.

[SCH 90] SCHIEL, Ulrich; MISTRIK, Ivan. Using Object-Oriented Analysis and Design for Integrated Systems. **Arbeitspapiere der GMD**, v.449, Jun. 1990.

[SHL 89] SHLAER, Sally; MELLOR, Stephen. An Object-Oriented Approach to Domain Analysis. **ACM SIGSOFT Software Engineering Notes**, New York, v.14, n.5, p.66-77, Jul. 1989.

[SHL 90] SHLAER, Sally; MELLOR, Stephen. **Análise de Sistemas Orientada para Objetos**. São Paulo: McGraw-Hill, 1990. 178p.

[SHL 92] SHLAER, Sally; MELLOR, Stephen. **Object Life Cycles: Modeling the World in States**. Englewood Cliffs: Yourdon Press, 1992.

[SEI 89] SEIDEWITZ, Ed. General Object-Oriented Software Development: Background and Experience. **The Journal of Systems and Software**, v.9, n.2, p.95-108, Feb. 1989.

[TEO 86] TEOREY, Toby; YANG, Dongqing; FRY, James. A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. **ACM Computing Surveys**, New York, v.18, n.2, Jun. 1986.

[WAL 92] WALTERS, Neal. Using Harel Statecharts to Model Object-Oriented Behavior. **ACM SIGSOFT Software Engineering Notes**, New York, v.17, n.4, p.28-31, Oct. 1992.

[WAR 85] WARD, Paul; MELLOR, Stephen. **Structured Devlopment for Real-Time Systems Volume I, II & III**. Englewood Cliffs: Yourdon Press, 1985.

[WAS 90] WASSERMAN, Anthony; PIRCHER, Peter; MULLER, Robert. The Object-Oriented Structured Design Notation for Software Design Representation. **IEEE Computer**, Los Alamitos, v.23, n.3, p.50-63, Mar. 1990.

[WIL 90] WILSON, D. Class Diagrams: A Tool for Design, Documentation, and Teaching. **Journal of Object-Oriented Programming**, New York, p.38-44, Jan./Feb. 1990.

[WIR 90] WIRFS-BROCK, Rebecca; JOHNSON, Ralph. Surveying Current Research in Object-Oriented Design. **Communications of the ACM**, New York, v.33, n.9, p.104-124, Sep. 1990.

[WIR 90a] WIRFS-BROCK, Rebecca; WILKERSON, B.; WIENER, L. **Designing Object-Oriented Software**. Englewood Cliffs: Prentice Hall, 1990.

[YAD 88] YADAV, Surya; BRAVOCO, Ralph; CHATFIELD, Akemi; RAJKUMAR, T. Comparison of Analysis Techniques for Information Requirement Determination. **Communications of the ACM**, New York, v.31, n.9, p.1090-1097, Sep. 1988.

[YOU 90] YOURDON, Edward. **Análise Estruturada Moderna**. Rio de Janeiro: Campus, 1990. 836p.

[YOU 90a] YOURDON, Edward. Auld Lang Syne. **Byte**, Peterborough, v.15, n.10, p.257-264, Oct. 1990.