

Banco de dados Orientado a Objetos

Arquivo e Banco de Dados

Índice

1. [Introdução](#)
 2. [Conceitos Básico](#)
 3. [Modelagem e Projeto para BDOO's](#)
 4. [Persistência em BDOO's](#)
 5. [Transações, Concorrência, Recuperação, Versionamento de BDOO's](#)
 6. [Conclusão e Bibliografia](#)
-

Banco De Dados Orientado a Objetos

1.0 Introdução

Hoje, o banco de dados orientados a objeto é um fator emergente que integra banco de dados e a tecnologia de orientação a objetos. Por um lado, a necessidade de realizar manipulações complexas para os banco de dados existentes e uma nova geração de aplicações de banco de dados geralmente requisitam mais diretamente um banco de dados orientado a objeto. Por outro lado, aplicações de linguagens orientadas a objeto e sistemas estão exigindo capacidades de banco de dados, tais como continuidade, simultaneidade e transações, dos seus ambientes. Estas necessidades estão levando à criação de sistemas poderosos, chamados *banco de dados orientados a objeto*.

Os bancos de dados orientados a objeto iniciaram-se primeiramente em projetos de pesquisa nas universidade e centros de pesquisa. Em meados dos anos 80, eles começaram a se tornar produtos comercialmente viáveis. Hoje, eles são mais de 25 produtos no mercado.

[Índice Próximo](#)

Banco de Dados Orientado a Objetos

2.0 Conceitos Básicos

Sistemas de Gerenciamento de Banco de Dados Orientado a Objetos

O desenvolvimento dos Sistemas de Gerenciamento de Banco de Dados Orientado a Objetos (SGBDOO) teve origem na combinação de idéias dos modelos de dados tradicionais e de linguagens de programação orientada a objetos.

No SGBDOO, a noção de objeto é usada no nível lógico e possui características não encontradas nas linguagens de programação tradicionais, como operadores de manipulação de estruturas, gerenciamento de armazenamento, tratamento de integridade e persistência dos dados.

Os modelos de dados orientados a objetos tem um papel importante nos SGBDs porque, em primeiro lugar, são mais adequados para o tratamento de objetos complexos (textos, gráficos, imagens) e dinâmicos (programas, simulações). Depois, por possuírem maior naturalidade conceitual e, finalmente, por estarem em consonância com fortes tendências em linguagens de programação e engenharia de software. O casamento entre as linguagens de programação e banco de dados é um dos problemas que estão sendo tratados de forma mais adequada no contexto de orientação a objetos.

Apresenta-se adiante os conceitos básicos de modelos de dados e SGBDs orientados a objetos.

Modelos de Dados Orientados a Objetos

Superficialmente, pode-se dizer que orientação a objetos corresponde à organização de sistemas como uma coleção de objetos que integram estruturas de dados e comportamento. Além desta noção básica, a abordagem inclui um certo número de conceitos, princípios e mecanismos que a diferenciam das demais. Seus principais conceitos são apresentados em seguida.

Abstração

É a consideração apenas das propriedades comuns de um conjunto de objetos, omitindo os detalhes, utilizada com frequência na definição de valores similares e na formação de um tipo a partir de outro, em diferentes níveis de abstração. O uso de abstrações permite a geração de tipos baseada em hierarquias de tipos e de relacionamentos.

Os principais conceitos de abstração utilizados em banco de dados são generalização e agregação. A

generalização corresponde à associação "é um" onde, a partir de propriedades comuns de diferentes entidades, é criada uma outra entidade. O processo inverso é a especialização. A agregação corresponde a associação "parte de".

Objeto

Os objetos são abstrações de dados do mundo real, com uma interface de nomes de operações e um estado local que permanece oculto. As abstrações da representação e das operações são ambas suportadas no modelo de dados orientado a objetos, ou seja, são incorporadas as noções de estruturas de dados e de comportamento. Um objeto tem um estado interno descrito por atributos que podem apenas ser acessados ou modificados através de operações definidas pelo criador do objeto. Um objeto individual é chamado de instância ou ocorrência de objeto. A parte estrutural de um objeto (em banco de dados) é similar à noção de entidade no modelo Entidade-Relacionamento.

Identidade de Objeto

Num modelo com identidade de objetos, estes têm existência independente de seus valores correntes e dos endereços de armazenamento físico. A identidade do objeto é geralmente gerada pelo sistema. A impossibilidade de garantir a identificação de objetos exclusivamente através de suas propriedades estruturais e comportamentais motivou a definição de identificadores únicos de objetos, que persistem no tempo de forma independente ao estado interno do objeto.

A identidade de objetos elimina as anomalias de atualização e de integridade referencial, uma vez que a atualização de um objeto será automaticamente refletida nos objetos que o referenciam e que o identificador de um objeto não tem seu valor alterado.

Objetos Complexos

Os objetos complexos são formados por construtores (conjuntos, listas, tuplas, registros, coleções, arrays) aplicados a objetos simples (inteiros, booleanos, strings). Nos modelos orientados a objetos, os construtores são em geral ortogonais, isto é, qualquer construtor pode ser aplicado a qualquer objeto. No modelo relacional este não é o caso, visto que só é possível aplicar o construtor de conjuntos às tuplas e o construtor de registro a valores atômicos.

A manutenção de objetos complexos, independente de sua composição, requer a definição de operadores apropriados para sua manipulação como um todo, e transitivos para seus componentes. Exemplos destas operações são: a atualização ou remoção de um objeto e cópia profunda ou rasa.

Encapsulamento

O encapsulamento possibilita a distinção entre a especificação e a implementação das operações de um objeto, além de prover a modularidade que permite uma melhor estruturação das aplicações ditas complexas, bem como a segurança dentro do sistema. Em banco de dados se diz que um objeto está encapsulado quando o estado é oculto ao usuário e o objeto pode ser consultado e modificado exclusivamente por meio das operações a ele associadas.

Existe uma certa discussão sobre as consultas em banco de dados quando está incorporada a noção de encapsulamento: Deve-se tornar visível apenas as operações e deixar ocultos os dados e as implementações ? É interessante relaxar o encapsulamento apenas para as consultas ? Como deve ser realizada a otimização de consultas em SGBDOO com encapsulamentos ?

Tipo de Objetos

O tipo de objeto pode ser visto como a descrição ou especificação de objetos. Um tipo possui duas partes, interface (visível para o usuário do tipo) e implementação (visível só para o usuário construtor do tipo).

Existem várias vantagens em se ter um sistema de tipos em um modelo de dados. Além de modularidade e segurança, do ponto de vista da evolução do sistema os tipos são especificações do comportamento que podem ser compostos e modificados incrementalmente, para formar novas especificações.

Classes

Um conjunto de objetos que possui o mesmo tipo (atributos, relacionamentos, operações) pode ser agrupado para formar uma classe. A noção de classe é associada ao tempo de execução, podendo ser vista como uma representação por extensão, enquanto que o tipo é uma representação intencional. Cada classe tem um tipo associado, o qual especifica a estrutura e o comportamento de seus objetos. Assim, a extensão da classe denota o conjunto dos objetos atualmente existentes na classe e o tipo provê a estrutura destes objetos.

Herança

Herança é um mecanismo que permite ao usuário definir tipos de forma incremental, por refinamento de outros já existentes, permitindo composição de tipos em que as propriedades de um ou mais tipos são reutilizadas na definição de um novo tipo. De fato, ela corresponde a transferência de propriedades estruturais e de comportamento de uma classe para suas subclasses.

As principais vantagens de herança são prover uma maior expressividade na modelagem dos dados, facilitar a reusabilidade de objetos e definir classes por refinamento, podendo fatorar especificações e implementações como na adaptação de métodos gerais para casos particulares, redefinindo-os para estes, e simplificando a evolução e a reusabilidade de esquemas de banco de dados.

Tipos de Herança

Os dois tipos de herança, simples e múltipla, são descritos a seguir:

- **Herança Simples:** Na herança simples um certo tipo pode ter apenas um supertipo, da mesma forma uma subclasse só herda diretamente de uma única classe. Podemos classificar esta herança em quatro subtipos: de substituição, de inclusão, de restrição e de especialização.
- **Herança Múltipla:** Nesta herança um tipo pode ter supertipos e os mesmos refinamentos de herança simples. Há basicamente dois tipos de conflitos referentes à herança múltipla: entre o tipo e o supertipo e entre múltiplos supertipos. O primeiro pode ser resolvido dando-se prioridade

à definição presente no tipo, e não a no supertipo. Com os conflitos entre múltiplos supertipos, como uma resolução por default pode causar heranças não desejadas, a abordagem mais segura é baseada na requisição explícita da intervenção do usuário.

Métodos e Mensagens

Um método, em relação a um objeto, corresponde ao comportamento dos objetos, implementando uma operação associada a uma ou mais classes, de forma similar aos códigos dos procedimentos usados em linguagens de programação tradicionais, que manipula o objeto ou parte deste. Cada objeto tem um certo número de operações para ele definida. Para cada operação pode-se ter um ou mais métodos de implementação associados.

As mensagens são a forma mais usada para se ativar os métodos. Num SGBDOO os objetos se comunicam e são ativados através de mensagens enviadas entre eles.

Polimorfismo

Em sistemas polimórficos uma mesma operação pode se comportar de diferentes formas em classes distintas. Como exemplo temos o operação print que será implementada de forma diferente se o objeto correspondente for um texto ou uma imagem: dependendo do objeto teremos um tipo de impressão. Tem-se também polimorfismo quando ocorre a passagem de diferentes tipos de objetos como parâmetros enviados a outros objetos

Um mesmo nome pode ser usado por mais de uma operação definida sobre diferentes objetos, o que caracteriza uma sobrecarga (overloading). A redefinição do operador para cada um dos tipos de objetos definidos caracteriza uma sobreposição (overriding). As operações são ligadas aos programas em tempo de execução caracterizando o acoplamento tardio ou late binding.

Outros conceitos

Finalmente há duas propriedades fundamentais para a construção de um SGBDOO: extensibilidade e completude computacional. A primeira garante que o conjunto de tipos oferecidos pelo sistema permite a definição de novos tipos e não há distinção entre os tipos pré-definidos e os definidos pelo usuário. A segunda implica que a linguagem de manipulação de um banco de dados orientado a objetos pode exprimir qualquer função computacional.

[Índice](#) [Anterior](#) [Próximo](#)

Banco de Dados Orientado a Objetos

6.0 Conclusão

O Banco de Dados entrou no mercado em meados da década de 80. Atualmente há mais de dez empresas de banco de dados orientado a objetos vendendo mais de 25 produtos de banco de dados orientado a objetos. Além disso, os banco de dados relacionais estão incorporando recursos orientados a objeto em seus produtos da próxima geração. Essas duas tendências devem prosseguir. Ambas proporcionam poderosos recursos de modelagem orientada a objeto aos desenvolvedores de aplicação.

Os BDOO's constituem uma importante etapa na evolução dos Sistemas gerenciadores de banco de dados. Além disso, embora os BDOO's sejam normalmente caracterizados como aqueles cujos recursos satisfazem às exigências das aplicações de banco de dados "avançadas", todas as aplicações de banco de dados podem se beneficiar dos poderosos conceito de modelagem dos BDOO's.

Num futuro próximo, muitos usuários se beneficiarão dos produtos que fornecem um modelo orientado a objetos com persistência e outros recursos de banco de dados. A ênfase será no benefício da orientação a objeto e o BDOO's para o usuário final. Tanto os desenvolvedores como os usuários finais se beneficiarão dos BDOO's. Todavia, há alguns desafios e questões em aberto, principalmente na área de padrões e modelos ajustados.

Em um mundo cujos problemas parecem ser agravadas com o tempo, uma simplificação de seus "modelos" talvez seja bem vinda. Os bancos de dados e os servidores de Banco de dados serão definitivamente a parte principal da computação à medida que caminhamos para o século XXI. Independente de ser como extensões de banco de dados relacionais ou como produtos e empresas de SGBDOO "novéis", espera-se que uma percentagem substancial das tecnologias de SGBD seja orientada a objetos.

Bibliografia

Khoshafian, Strag - BANCO DE DADOS ORIENTADOS A OBJETOS - IBPI press, WILEY

Internet

<http://www.rhein-neckar.de/~cetus/software.html>

<http://www.nce.ufrj.br/~marciosd/pfinal>

<http://cogae.pucsp.br/~fea/bjsucesu.htm>

[Índice Anterior](#)

Banco de Dados Orientado a Objetos

4.0 Persistência em Banco de Dados Orientado a Objeto.

O termo persistência é raramente utilizado no contexto de banco de dados. Preferencialmente, o termo usado é banco de dados, que conota o espaço de objeto resiliente, concorrentemente compartilhado. A função de um sistema de gerenciamento de banco de dados é permitir o acesso e a atualização simultâneos de bancos de dados persistentes. A fim de garantir a persistência dos dados a longo prazo, os sistemas de gerenciamento de banco de dados utilizam várias estratégias de recuperação em caso de falhas na transação, no sistema ou no meio.

Há uma relação fundamental entre o compartilhamento e a persistência simultâneos em banco de dados. As atualizações de transação devem persistir, mas, como o banco de dados persistentes é ao mesmo tempo acessado e atualizado, o sistema de gerenciamento de banco de dados deve preocupar-se com a coerência dos objetos de dados persistentes. Isso normalmente é obtido por meio de estratégias de controle e recuperação concorrentes.

Níveis de Persistência

Os dados manipulados por um banco de dados orientado a objeto podem ser transientes ou persistente. Os dados transientes só são validos dentro de um programa ou transação, eles se perdem quando o programa ou a transação termina. Os dados persistentes, por outro lado, são armazenados fora do contexto de um programa e assim sobrevivem a varias invocações de programas.

Dados persistentes normalmente consistem nos bancos de dados compartilhado, acessados e atualizados através de transações. Por exemplo, banco de dados pessoais, banco de dados de inventário e banco de dados de vendedores, contas ou itens, todos contem dados persistentes. No entanto, há vários níveis de persistência. Os objetos menos persistentes são aqueles criados e destruídos em procedures. Depois, há os objetos que persistentem dentro do espaço de trabalho de uma transação, mas que são invalidados quando a transação termina. As transações são normalmente executadas dentro de uma sessão. O usuário estabelece seu login e define diferentes parâmetros ambientais dentro de um sessão, como caminhos, opções de exibição, janelas, etc. Se o sistema suportar o multiprocessamento, várias transações poderão estar ativas dentro da mesma sessão de usuário ao mesmo tempo. Todas estas transações compartilharão os objetos da sessão. No entanto, quando o usuário terminar a sessão, os objetos da sessão serão invalidados. O único tipo de objeto que persiste através das transações são objetos permanentes normalmente compartilhados por vários usuários. Esses objetos persistem através de transações, instabilizações de sistema e ate de meio. Tecnicamente, esses são os objetos recuperáveis do banco de dados.

[Índice](#) [Anterior](#) [Próximo](#)

Banco de Dados Orientado a Objetos

5.0 Transações, Concorrência, Recuperação e Vercionamento de BDOO's

Transações

Uma transação é um programa executado inteiramente ou então não executado. As transações devem mapear bancos de dados de um estado coerente para outro. Para manter a coerência, as transações devem passar pelo teste ACID: Atomicidade, coerência, isolamento, e durabilidade.

Atomicidade

Como uma transação é executada inteiramente ou então não é executada, ou a seqüência completa de operações é aplicada ao banco de dados ou então nenhuma. Este recurso chama-se de Atomicidade; as transações são atômicas.

Coerência

Diz-se que o banco de dados é coerente se todas as suas restrições de integridade são satisfeitas. Pressupõe-se que na execução de uma transação, na ausência de interferência de outras transações concorrentes, o banco de dados seja levado de um estado coerente para outro.

Isolamento

Como as transações são executadas concorrentemente no mesmo banco de dados, elas devem ser *isoladas* das outras operações. Do contrário, a operação intercalada de transações concorrente pode levar a anomalias. Assim, os SGBD suportam isolamento, que fornece segurança contra interferências entre as transações concorrentes.

Durabilidade

A durabilidade está relacionada à capacidade do SGBD de se recuperar de falhas no sistema e no meio. As atualizações de uma transação efetivada devem ser preservadas e registradas em algum meio durável. Deve-se manter redundância suficiente para que se reconstrua um banco de dados coerente.

Transações aninhadas

As transações de aplicações de banco de dados orientadas a objetos são normalmente mais demoradas que as de aplicação comerciais convencionais. Longa duração das transações em aplicações avançadas é uma característica das aplicações de banco de dados da próxima geração. Varias estratégias relacionadas á longa duração das transações foram propostas na pesquisa de banco de dados. Algumas estratégias influenciaram as implementações de banco de dados orientado a objetos.

As transações aninhadas são utilizadas para resolver alguns problemas associados as transações de longa duração. Um modelo de transação aninhada pode conter subtransações, também chamadas de transações-filhas. Em uma transação aninhada, todas as transações-filhas devem ser efetivadas para que a transação de alto nível se efetive. Cada subtransação deve ser concluída ou abortada. Também, em aplicações avançadas, as tarefas normalmente envolve vários usuários. As transações em cooperação são utilizadas para suportar essas tarefas em conjunto.

Concorrência

Vários algoritmos de controle podem ser usados para garantir a capacidade de serialização das transações e a coerência do banco de dados. O mais notável deles é o *bloqueio*. Nos bancos de dados orientados a objeto, o bloqueio pode ser associado a vários grânulos que são manipulados pelos usuários, incluindo classes, instancias e objetos complexos.

Nos bancos de dados orientados a objeto, há dois aspectos de bloqueio que são relevantes para o compartilhamento concorrente de objetos:

Bloqueio de Hierarquia de classe: As classes nos bancos de dados orientados a objeto são organizadas em hierarquias de herança, de modo que cada classe da hierarquia tenha uma extensão ou instancia preexistente. Por isso é importante fornecer bloqueio de granularidade a essas estruturas. Por exemplo, uma superclasse poderia bloquear implicitamente todas as subclasses no mesmo modo de bloqueio. As subclasses incluem os descendentes diretos da superclasse e os descendentes de suas subclasses.

Bloqueio de Objeto complexo: Os bancos de dados orientados a objetos contêm objetos que podem referenciar ou incorporar outros objetos. Além disso, alguns objetos são "valores", enquanto outros possuem identidade. Para otimizar a concorrência na presença de modelos que envolvam objetos complexos, foram analisados vários esquemas de bloqueio de "objetos compostos" ou de "objetos dependentes" para objetos complexos.

Recuperação

A confiabilidade e a grata recuperação de falhas são importantes recursos de um sistema de gerenciamento de banco de dados. O *gerenciador de recuperação* é o modulo que administras as técnicas de recuperação dessas falhas. Os três importantes tipos de falhas que são responsabilidade do

gerenciador de recuperação são: as *falhas de transação*, as *falhas no sistema*, as *falhas no meio*. Uma das estruturas mais utilizadas para o gerenciamento de recuperação é o log. O log é utilizado para registrar e armazenar as imagens anteriores e posteriores dos objetos atualizados. A imagem anterior é o estado do objeto antes da atualização da transação, e a imagem posterior é o estado do objeto após a atualização da transação. Quase todos os bancos de dados orientados a objeto suportam a recuperação. A maioria dos SGBDOO utiliza o logging para a recuperação do banco de dados a um estado coerente. Alguns utilizam a duplicação ou espelhamento de dados.

Vercionamento

O acesso a estados anteriores ou a estados alterados de objetos é parte inerente de muitas aplicações. Ele é obtido por meio de várias versões do mesmo objeto. O gerenciamento de versão em um banco de dados orientados a objeto consiste em ferramentas e construções que automatizam ou simplificam a construção e a organização de versões ou configurações. Sem essas ferramentas, caberia ao usuário organizar e manter as versões.

Podemos considerar a configuração como um grupo de objetos tratados como uma unidade para bloqueio e Vercionamento. Os objetos individuais dentro da configuração podem sofrer modificações, de modo que cada objeto pode Ter um histórico das versões. Vários objetos dentro da configuração são atualizados em momentos diferentes e não necessariamente na mesma frequência.

[Índice](#) [Anterior](#) [Próximo](#)

Banco de Dados Orientado a Objetos

3.0 Modelagem e Projeto para BDOO's

ANÁLISE ORIENTADA A OBJETOS

Diretamente derivada dos conceitos da programação e do projeto orientado a objeto, a análise orientada a objeto é certamente a mais nova das abordagens de análise de sistemas. Baseada na decomposição do sistema de acordo com os objetos (entidades de dados) manipulados por este, esta abordagem oferece como principais benefícios os seguintes fatores:

- a) Mantém a modelagem do sistema e, conseqüentemente, a automação do mesmo o mais próximo possível de uma visão conceitual do mundo real.
- b) Baseia a decomposição e modelagem do sistema nos dados, que é o elemento mais estável de todos aqueles que compõem um sistema de informação.
- c) De todas as abordagens de análise conhecidas até o momento, a análise orientada a objetos é, certamente, a que oferece maior transparência na passagem da análise (modelo essencial) para o projeto (modelo de implementação). Isso porque, se for seguida a técnica de projeto orientado a objeto, a passagem de um modelo para o outro será feita através da introdução de detalhes, não requerendo uma reorganização do modelo, como é o caso na passagem de análise estruturada para o projeto estruturado.

É interessante que façamos a definição de Classes e Objetos, (9):

- a) Objeto: "Uma abstração de alguma coisa em um domínio de problema, refletindo as capacidades de um sistema de manter informações sobre sí, interagir com sí, ou ambos, um encapsulamento de Valores, de Atributos e seus Serviços exclusivos".
- b) Classe: "Uma coleção de objetos que podem ser descritos com o mesmo Atributo ou Serviço".

Análise Orientada a Objeto, uma abordagem passo a passo

Pode-se perguntar: Porque não usar as técnicas de análise já estabelecidas, como a análise estruturada. A evolução da análise estruturada nos mostra que há uma série de desenvolvimentos, a programação estruturada é precedida pelo projeto estruturado, que por sua vez é precedido pela análise estruturada. A análise e o projeto estruturado são construídos no topo de noções básicas da programação estruturada tradicional, incluindo a separação de dados e processos.

Algumas noções básicas de programação orientada a objetos são diferentes da programação tradicional.

Por instância, objetos encapsulam ambos comportamento e estado, enquanto as linguagens tradicionais de programação deliberadamente mantêm uma separação.

A seguir apresentaremos uma abordagem de desenvolvimento de sistemas orientados a objetos, numa abordagem passo a passo, (3):

OBA (Object Behavior Analysis) provém um modelo conceitual para um primeiro estágio na criação de uma aplicação orientada a objeto. Usando esta abordagem como um passo positivo no processo de construção, com sucesso, de um sistema flexível orientado a objeto.

A OBA tenta responder questões que a análise tradicional de sistemas não consegue responder, e isto de uma maneira natural, encorajando a interação entre as diversas fases do desenvolvimento e com o auxílio da prototipação rápida especialmente nas fases de análise e projeto.

Uma implicação desta abordagem é que analisar, projetar e implementar, não do tipo, "vamos fazer certo da primeira vez". Na prática geralmente isso é impossível, ter a informação que se necessita para uma análise completa, muito menos interpretar e representar todas as coisas corretamente sem interações. Interações controladas contribuem para clarificar e tornar mais completas as especificações e projeto.

OBA consiste de uma compreensão da aplicação e identificação dos comportamentos iniciais, definindo os objetos que exibem esses comportamentos, classificando os objetos, identificando os relacionamentos entre eles e modelando seus ciclos de vida.

a) Etapa 1: Identificando os comportamentos

Para compreender o que a aplicação fará, primeiro entrevista-se o usuário para a aplicação pretendida observando então a ação para ver o que ela fará, quem ou o quê interagirá com o sistema, em que ordem ocorrerá, e quais as diferentes ações tomadas. O ponto principal é uma lista dos desejos necessários para o sistema analisado, o mais alto nível do comportamento define as **responsabilidades** do sistema.

A saída desta etapa é uma lista dos comportamentos desejados do sistema e uma lista de suas propriedades visíveis, bem como de "scripts".

b) Etapa 2: Definindo os objetos

Uma vez definidos os comportamentos iniciais necessita-se determinar a sua performance. Encontrar os objetos é um passo para compreender a sua performance. Imediatamente vê-se quem ou o que é responsável por um comportamento particular.

Cartões de modelagem, usados nesta etapa, são pequenos cartões indexados com os termos: "nome", "responsabilidade" e "colaboradores", escrito neles. Nesses cartões escreve-se o comportamento do sistema (responsabilidades), o que ou quem é responsável por um comportamento em particular, tão bem quanto para padrões de comportamento (nome) e quem ou o que este agente responsável interage (colaboradores).

Os objetos de aplicação são compostos desses objetos concretos, conceitos, processos e eventos que exibem comportamentos significativos.

A saída da etapa 2 é uma lista de objetos, os quais relatados em grupos de comportamento e propriedades visíveis. Pode ser necessário fazer esta etapa mais de uma vez para atualizar a lista de objetos (p. ex.: adicionar uma classe de objetos).

c) Etapa 3: Classificando os objetos

Neste caso, classificação significa agrupar objetos de acordo com alguma similaridade de comportamento (função) e estado (forma). Classificar objetos primariamente pelo comportamento é a chave da OBA. Para isso, deve ser visto comportamentos similares em diferentes objetos.

A saída da etapa 3 é um diagrama de relacionamento hierárquico entre os objetos, baseados na forma abstrata usando o critério de comportamento similar. Uma importante tarefa no mundo da orientação a objeto é distinguir o abstrato a partir do específico. OBA inicia o projeto com o pé direito, por auxiliar na localização de conceitos que serão usados para ambas classes, abstrato e concreto durante o projeto e implementação.

d) Etapa 4: Identificando os relacionamentos

A etapa 4 envolve um desenho de um esboço preliminar dos objetos em relação uns com os outros. Usando esse esboço, tão bem quanto um "script" e os cartões de modelagem, pode -se criar uma tabela que clarifica os relacionamentos que cada objeto tem uns com os outros. Para OBA, necessita-se obter uma descrição, a mais clara possível, em termos de linguagem natural, dos relacionamentos entre os objetos. Então simplifica-se essas descrições de linguagens. De outro modo, reduzindo uma descrição simplificada em linguagem natural para os subconjuntos dos relacionamentos implementáveis em OOL, é parte da fase de projeto e não da fase de análise.

e) Etapa 5: Modelando o Processo

Necessita-se para determinar quais objetos iniciam atividades e identificar qual a sequência das atividades, para isso são usados os "scripts" desenvolvidos na etapa 1 para iniciar o modelo de aplicação.

Pode-se especificar o ciclo de vida dos objetos e seus "status" em diferentes partes do ciclo.

f) A Análise Final:

O resultado da OBA são as especificações das necessidades, escrita em termos de comportamento requerido, que delinea os objetos primários como eles são organizados. Esta especificação inclui objetos superordenados a partir de um agrupamento por comportamento e objetos subordinados.

O comportamento primário e estado da cada objeto é listado, bem como seus relacionamentos dos objetos e as sequências de atividades. Os "scripts" produzidos na OBA são especialmente usados no detalhamento de como trabalhará a interface com o usuário.

Com o uso da OBA para análise de um sistema orientado a objeto, obtém-se uma especificação do comportamento que enfatiza os aspectos da alta reusabilidade do sistema, que é, os protocolos de comportamento e os agrupamentos hierárquicos dos objetos de acordo com o protocolo.

Uma análise cuidadosa, focando as abstrações comportamentais, promovem uma redução no código, por calçar o caminho para uma boa construção hierárquica e hereditariedade e o seu uso prudente pode ajudar a produzir uma clara e compreensível estrutura de aplicação orientada a objeto.

PROJETO ORIENTADO A OBJETOS (OOD)

Uma vez construído o Modelo Essencial do sistema, necessitamos construir o seu Modelo de Implementação, ou seja, o seu projeto físico. Para essa fase do ciclo de vida do sistema, assim como para as demais, existem diversos métodos alternativos, como o Projeto Estruturado, e o Projeto Orientado a Objetos.

Um problema existente no projeto estruturado é a sua taxa de subjetividade dos conceitos de modularização.

Alguns objetivos chave para o projeto orientado a objetos (9):

- a) Aumento da produtividade pelo aumento da manutenibilidade e ênfase na reusabilidade.
- b) Incremento da qualidade, por **ênfase no processo** de desenvolvimento de software, e não unicamente no produto final.
- c) Aumentar a manutenibilidade por separar intrinsecamente partes voláteis do sistema daquelas que são estáveis.

Já no Projeto Orientado a Objetos, temos alguns benefícios tais como:

- a) A arquitetura do sistema é desenvolvida a partir dos objetos por ele manipulados. Em um sistema de

informações, os objetos correspondem às entidades de dados utilizadas.

b) Para cada objeto são definidas as funções ou serviços que deverá permitir, direta ou indiretamente, o atendimento aos requisitos estabelecidos. Todos os serviços correspondentes a um mesmo objeto devem ser integrados e estruturados em um único módulo, estabelecendo-se assim a funcionalidade do objeto.

c) Um objeto, através de seus serviços, pode colaborar com outros objetos. Esta colaboração se processa através da troca de mensagens entre os objetos cliente e servidor.

a) O objeto pode também herdar algumas funcionalidades correspondentes a seu objeto "pai", o qual deve representar, conceitualmente, uma entidade mais genérica.

Entre as principais vantagens oferecidas pelo método do projeto orientado por objeto temos:

a) Fornece regras precisas para a modularização do sistema.

b) Desenvolvimento de uma arquitetura mais estável, uma vez que se estrutura com base no modelo de dados utilizado, o qual apresenta pouca vulnerabilidade a mudanças na funcionalidade do sistema.

c) Alto grau de reusabilidade das rotinas desenvolvidas.

d) Maior facilidade de transição da fase de análise para a fase de projeto.

PROTOTIPAÇÃO ORIENTADA A OBJETO

A **prototipação orientada a objeto** (8) é baseada em abstração de dados e hereditariedade. Objetos encapsulam o dado em sistemas de prototipação servindo como base para o projeto e a implementação. Desde que o dado na implementação é geralmente mais estável que as etapas do processo, isto conduz a descrição do sistema que é mais fácil para ser modificado que através da abstração procedural. Hereditariedade ajuda a reduzir o trabalho envolvido na construção do sistema por inclusão de aspectos comuns de código em diferentes contextos sem a repetição explícita de detalhes. Objetos também provem componentes convenientes para código reusavel, processamento paralelo e controle de versões, desse modo, as abordagens orientadas a objeto fazem protótipos mais flexíveis e facilmente automatizáveis.

O OOD e a prototipação (9) tem vínculos em comum pois; é uma maneira natural de trabalhar, experimenta-se a interface humana, para a clarificação dos requisitos procurados, para a praticidade do projeto, para a entrega, funcionalmente, tão cedo quanto possível.

CONCLUSÕES

A orientação a objetos (2) é uma mudança cultural, muito mais do que uma mudança tecnológica, ela é um paradigma, uma revolução industrial do software baseada na reusabilidade e intercambialidade entre as partes que alterará o universo de software tão fortemente quanto a revolução industrial alterou a manufatura.

O paradigma muda das linguagens procedurais para as linguagens orientadas a objeto, com o direcionamento para ambientes de alto nível como o Smalltalk.

Do mesmo modo as metodologias mudam das estruturadas para as orientadas a objetos, não podendo haver uma continuidade nas técnicas estruturadas existentes, confirmando (1), "os métodos estruturados Não podem ser objetificados".

Grandes organizações mudam lentamente, velhas metodologias, quando há, são difíceis de mudar o no entanto se requer uma mudança radical na maneira de como sistemas serão desenvolvidos a partir de agora, isto é, orientados a objeto e com auxílio de CASE.

Os anos 90 serão de gradual aceitação da orientação a objetos e não podendo tomar por base antigas metodologias.

Orientação a objetos promete muito mais do que "apontar e clicar" a interface de janelas, e com a popularização das arquiteturas abertas, redes distribuídas, CASE e ambientes gráficos o seu destino está definido.

[Índice](#) [Anterior](#) [Próximo](#)