JANE MONTIN PELLIM

Proposta de Migração da Metodologia Análise Estruturada para Orientação a Objetos

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de MBA em Engenharia

São Paulo
2003

JANE MONTIN PELLIM

Proposta de Migração da Metodologia Análise Estruturada para Orientação a Objetos

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de MBA em Engenharia.

Área de Concentração:
Engenharia de Software

Orientador:
Prof. Doutor
Reginaldo Arakaki

São Paulo
2003

À minha mãe pelo espírito de perseverança
que me inspira.

AGRADECIMENTOS

A Deus.

A todos os colegas e professores, em especial meu orientador Prof. Doutor Reginaldo Arakaki pela sugestão do tema e pelas diretrizes seguras.

À empresa Elucid Solutions S/A pela oportunidade na participação de projetos que proveram material para o Estudo de Caso dessa dissertação.

RESUMO

O objetivo deste trabalho é apresentar a elaboração, a implementação e a repercussão de um estudo no qual foram apresentadas técnicas para profissionais, da área de informática, interessados em migrar da Análise Estruturada para Orientação a Objetos, ao desenvolver projetos de software. Partindo da constatação do rápido avanço tecnológico e de depoimentos de atuantes na área, a investigação procurou levantar os fundamentos conceituais, métodos e uso de modelos que utilizam as técnicas mais modernas disponíveis atualmente. Assim, considerando a natureza qualitativa da investigação, o estudo de caso foi a abordagem metodológica adotada, o qual realizou-se em uma empresa particular, do segmento de informática, sediada no estado de São Paulo, e que materializa toda a teoria expressa nesse trabalho. Os resultados revelaram que, apesar do potencial interativo e realista da Tecnologia Orientada a Objetos, muitos profissionais ainda não lançam mão de tal recurso por, possivelmente, considerá-lo mais complexo do que realmente é, e por, infelizmente, acreditar que seja totalmente desvinculado da Tecnologia de Análise Estruturada. Desta forma, este texto tem a pretensão de desmistificar esses pensamentos e transmitir experiência da prática de projeto orientado a objeto a profissionais que queiram aprendê-lo e utilizá-lo em seus projetos, elucidando esta nova tecnologia e facilitando o período de transição em que muitos possam se encontrar.

ABSTRACT

The aim of this study is to present the design, the development and the results of an investigation in which were presented some techniques for professionals from Computer field interested in moving from Structured Analysis to Object-Oriented, during the development of software projects. By taking the fast technology improvements and expertise statements into account, this study tried to detect the conceptual foundations, methods and the usage of models that are current available. Due to the qualitative nature of the investigation, case study was selected as the methodological approach to be used, which was held in a private computer company situated in São Paulo State. The results revealed that, in spite of the interactive and realistic features of oriented-object technology, many professionals do not take advantages of this resource yet because they, probably, believe is too complex as well as they, unfortunately, can not see that the grounds of both technologies do not diverge. Furthermore, the development of this study can potentially contribute to the familiarization of this technique and make smoother the transition period in which many experts can be.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

DER	**- Diagrama Entidade-Relacionamento**
DFD	**- Diagrama de Fluxo de Dados**
DHF	**- Diagrama Hierárquico de Funções**
EDI	**- Eletronic Data Interchange**
IHM	**- Interface Homem-Máquina**
JDBC	**- JAVA Database Connectivity**
JDK	**- JAVA Development Kit**
JRE	**- JAVA Runtime**
JSP	**- JAVA Server Pages**
MER	**- Modelo Entidade-Relacionamento**
ODBMS	**- Object DataBase Management System**
OO	**- Orientação a Objeto**
OMG	**- Object Management Group**
UML	**- Unified Modeling Language**

1 INTRODUÇÃO

Este capítulo apresenta os subsídios necessários, que definem o contexto geral, da monografia intitulada de "Proposta de Migração da Metodologia Análise Estrutura para a Orientação a Objetos".

O contexto geral deste capítulo, está composto de tópicos referentes aos objetivos, abrangência do trabalho, motivação, justificativa, histórico da pesquisa e a metodologia utilizada.

Considerações Iniciais

Com o advento da crise do software, tenta-se, a pelo menos 30 anos, aplicar conceitos e práticas de engenharia no desenvolvimento e manutenção de *software* .

Os problemas mais freqüentemente encontrados eram:

- Dificuldade de o software acompanhar a evolução do hardware;
- Atraso nos prazos de entrega do software;
- Falta de flexibilidade em manutenção do software para contemplar futuras mudanças em requisitos;
- Baixa participação dos clientes/usuários durante o processo de desenvolvimento;

Para tentar solucionar esses problemas surgiu a Engenharia de Software, que é a aplicação de princípios, métodos, técnicas, ferramentas, processos e gerenciamento, visando o desenvolvimento racional e econômico de produtos de software.

O presente trabalho se preocupa com o item de metodologia de desenvolvimento, focando a Orientação a Objetos.

Para a melhoria dos produtos de software, pode-se atacar vários pontos acima expostos, tais como, processo, gerência. O presente trabalho procura dar a sua contribuição para amenizar os problemas acima expostos, através do estudo de metodologia, julgando ser a Orientação a Objetos, a mais útil e necessária atualmente.

Como muitos profissionais ainda trabalham com Análise Estruturada, esse é o público preferencial a quem se destina o trabalho.

1.1 Objetivo da Monografia

Este trabalho refere-se à investigação, elaboração e implementação de uma proposta de desenvolvimento de sistemas de informação, a partir da compreensão e aplicação correta do novo paradigma de Desenvolvimento de *Software*: a Tecnologia de Objetos.

A partir de um estudo comparativo entre a análise estruturada e a orientação a objetos, serão identificadas as diferenças entre essas técnicas, fazendo um paralelo entre as atividades exercidas nas etapas de desenvolvimento de sistemas utilizando uma metodologia e outra, explicando as mudanças que devem ocorrer na maneira de raciocinar para a utilização da nova técnica, que é pensar em objetos.

Partindo desse interesse, o trabalho relata e interpreta, através de um Estudo de Caso, uma experiência vivida em uma empresa particular da área de informática do estado de São Paulo, em relação ao desenvolvimento de uma funcionalidade básica de um sistema de informações modelo que utilizará essa Tecnologia de Desenvolvimento, permeado pelos princípios teóricos abordados nesta monografia.

A tecnologia emergente da OO, adotada amplamente pelas empresas de desenvolvimento de *software*, é a motivação da escolha do tema e o estudo mais aprofundado sobre essa tecnologia. Desta forma, o estudo de caso foi considerado a melhor abordagem metodológica, com uma perspectiva qualitativa de interpretação, tentando investigar a repercussão da OO através de uma coleta de vários instrumentos e métodos interconectados.

1.2 Abrangência

O escopo deste trabalho de monografia, estará restrito ao contexto de investigação, elaboração e implementação de uma proposta de metodologia para as fases de análise e projeto orientados a objeto.

A proposta de desenvolvimento aqui apresentada, com a relação aos artefatos produzidos em cada fase e a ordem em que são elaborados, é uma dentre uma série de alternativas existentes no mercado, sendo, portanto, uma sugestão a ser adotada.

É bom ressaltar, que o escopo desta monografia não engloba detalhes da fase de implementação do desenvolvimento de software.

1.3 Motivações para o uso da técnica Orientada a Objetos

A motivação para realizar este estudo originou-se, também, de declarações de grandes profissionais e autoridades em Engenharia de Software, a respeito da tecnologia de orientação a objetos, que explicam a sua crescente adoção. Abaixo são apresentados alguns trechos das declarações:

"Três aspectos foram relevantes para a ascensão da orientação a objetos, contrapondo-se aos métodos estruturados: a unificação de conceitos entre as fases de análise e programação, o grande potencial de reutilização do *software* e a facilidade de manutenção. A orientação a objetos também se apresentou com a esperança de suprir algumas das preocupações da indústria do software: a necessidade de criar softwares corporativos muito mais rapidamente, mais confiáveis e a um custo baixo. Os meios para conseguir essa proeza encontram-se nas características apresentadas pelo paradigma da orientação a objetos, o que leva a um salto quantitativo e qualitativo na produção do software." [Booch]

"... a orientação a objetos ... uma nova maneira de pensar os problemas utilizando modelos organizado as partir de conceitos do mundo real. O componente fundamental é o objeto que combina estrutura e comportamento em uma única entidade" [Rumbaugh]

"... a questão mais importante da revolução tecnológica baseada em objetos é a de reduzir a distância semântica que hoje existe entre os modelos utilizados nas linguagens de programação e nos sistemas gerenciadores de bancos de dados e os modelos conceituais que normalmente utilizamos quando pensamos sobre o mundo real." [OMG]

Martin (1994,p.58) defende que "à medida que os computadores se tornam mais complexos , os humanos não devem ter que pensar como computadores; ao contrário, os computadores devem ser construídos para pensarem como humanos.".

Outro aspecto importante é apontado por Melo(2002,p.7), neste depoimento: "Estamos vivendo uma era na qual precisamos mais do que entregar softwares no prazo, dentro do orçamento e sem falhas. Precisamos desenvolver novos softwares gastando menos tempo, economizando efetivamente .

Temos comprovado que a orientação a objetos vem propiciar à comunidade de desenvolvedores aumento de produtividade, diminuição do custo de

desenvolvimento e principalmente de manutenção, maior portabilidade e reutilização de código."

"A aplicação dos conceitos da orientação a objetos na análise de sistemas permitirá modelar a empresa ou as áreas da aplicação de uma forma mais natural, visto que os recursos a serem aplicados retratam com mais fidelidade o mundo real. A capacidade de retratar o universo pesquisado é moldado em diagramas que refletem exatamente o arranjo dos objetos no mundo real. A própria transição entre etapas na construção do software, aplicando-se o paradigma orientado a objetos, são refinações sucessivas com os mesmos conceitos e linguagem." [Tonsig, Sérgio Luiz]

Furlan (1998,p.12) reforça a naturalidade da técnica OO, ao afirmar que "um aspecto interessante da teoria de objetos é a sua característica intrínseca de analisar o mundo como ele é, permitindo organizar resultados de maneira mais fácil e natural. Procura representar o mundo como o vemos. A teoria de objetos espelha o mundo de forma mais simples do que modelos algorítmicos ou estruturados".

Por essas declarações, as vantagens da técnica OO para o desenvolvimento de sistemas e sua adoção crescente em corporações deverão ser observadas. Isto se deve ao cenário dinâmico dos negócios da atualidade, no qual as estruturas sistêmicas que lhes dão suporte precisam apresentar flexibilidade para acompanhar as mudanças.

1.4 Justificativa

A grande demanda de profissionais da Análise Estruturada em se aperfeiçoar tecnicamente e conhecer a nova técnica da Orientação a Objetos, que está atualmente em franca ascensão, cria a justificativa necessária para poder realizar esta monografia.

Através da apresentação da proposta de migração de metodologia, sugerida por esse trabalho, espera-se fortalecer a aceitação da técnica em Orientação a Objetos, especialmente entre os profissionais da Análise Estruturada.

Espera-se também, que essa monografia sirva de incentivo para novos trabalhos de pesquisa em diferentes níveis acadêmicos dentro da linha denominada Orientação a Objetos. Esta razão cria outra justificativa da monografia.

1.5 Metodologia

Para o desenvolvimento desta monografia, foi utilizada uma metodologia cujas fases são definidas a seguir:

- Pesquisa: fase que engloba a seleção das informações de caráter conceitual que fazem parte da fundamentação teórica que sustentará as fases seguintes deste trabalho;

- Estruturação da proposta de migração de metodologia: fase que abrange o desenvolvimento de uma sugestão de migração da metodologia de Análise Estruturada para a Orientação a Objetos, através da determinação das fases, procedimentos e artefatos produzidos durante um ciclo de desenvolvimento de software.

- Estudo de Caso: fase que engloba as atividades que colocam em prática a proposta apresentada no item anterior, consistindo na análise e modelagem de um pequeno módulo de um sistema de informações da área financeira, seguindo a proposta de migração de metodologia apresentada nesta monografia.

1.6 Histórico da Pesquisa

Este item tem por objetivo, apresentar a evolução da pesquisa sobre a migração de metodologia de Análise Estruturada para a Orientação a Objetos, realizada em uma empresa particular da área de informática do estado de São Paulo, tendo o acompanhamento e a supervisão do orientador desta monografia, o professor Doutor Reginaldo Arakaki da Escola Politécnica da Universidade de São Paulo, e que envolve o trabalho de monografia do autor.

A pesquisa inicia-se pelo interesse e apreciação do autor, pela nova técnica emergente no mercado, que é a Orientação a Objetos.

As disciplinas estudadas durante o curso de MBA em Engenharia de Software, intensificam e aguçam a curiosidade do autor em aplicar corretamente a nova técnica, decidindo pela opção de assunto da monografia em Orientação a Objetos.

Com a ajuda e supervisão do orientador dessa monografia, chegou-se à conclusão sobre um tema específico e promissor na área de Engenharia de Software, sobre métodos de desenvolvimento de *software*, que visa auxiliar profissionais que se encontram na mesma situação do autor da monografia, que desejam adotar uma proposta a ser seguida para uso da metodologia de Orientação a Objetos, fazendo um paralelo com atividades desempenhadas na metodologia de Análise Estruturada, para facilitar e agilizar essa migração.

1.7 Estrutura da Monografia

A estrutura da monografia está composta de cinco capítulos, cujos conteúdos serão definidos a seguir:

- Primeiro capítulo: nesse capítulo são apresentadas as razões que justificam o desenvolvimento desta monografia, assim como a definição dos objetivos que serão atingidos;

- Segundo capítulo: neste capítulo serão apresentados conceitos fundamentais sobre Orientação a Objetos;

- Terceiro capítulo: será apresentada a proposta de migração da metodologia de Análise Estrutura para a Orientação a Objetos;

- Quarto capítulo: apresenta o estudo de caso, que é o experimento que valida a proposta apresentada no terceiro capítulo;

- Quinto capítulo: apresenta as conclusões gerais da monografia, os trabalhos que poderão ser realizados no futuro dentro do contexto desta monografia, comentários finais e a contribuição acadêmica atingida.

2 FUNDAMENTOS CONCEITUAIS

2.1 Conceitos Fundamentais da Orientação a Objetos

Na tentativa de desmistificar a OO, serão apresentados, neste capítulo, os termos freqüentemente mencionados nessa área. Dada a naturalidade da técnica, sempre que possível, será feita uma analogia com o mundo real.

Todos esses conceitos, aqui apresentados, fazem parte da fundamentação teórica que sustentará o estudo de caso exposto no capítulo 4.

2.1.1 Mudança de Paradigma

A programação tradicional estruturada consiste em projetar, primeiramente, um conjunto de funções para resolver um problema, e depois se preocupar com a estrutura de dados. A POO **inverte** essa ordem e coloca as estruturas de dados em primeiro lugar, examinando depois os algoritmos que vão processar os dados.

Conforme Jandl Junior, " Essa nova forma de programar, embora mais complexa, é muito mais apropriada para o desenvolvimento de sistemas, pois permite tratar os problemas de forma semelhante ao nosso entendimento de mundo, em vez de dirigir a solução para as características de funcionamento dos computadores, tal como faz a tradicional programação procedural".

O desenvolvimento orientado a objeto é o inverso de metodologias orientadas à função. Nessas metodologias, a ênfase é dada à especificação e decomposição das funcionalidades do sistema. Embora pareça ser uma abordagem

mais direta, o sistema resultante torna-se mais frágil: se os requerimentos mudam, um sistema baseado na decomposição de funcionalidades pode exigir uma enorme reestruturação.

Ao contrário, a abordagem orientada a objeto focaliza primeiro a identificação de objetos no domínio da aplicação, e depois encaixa procedimentos à sua volta. Embora a OO pareça mais indireta, *softwares* nesta abordagem mantêm-se melhor à medida que os requerimentos evoluem, pois ela é baseada na estrutura sob o domínio da própria aplicação, e não nos instáveis requerimentos funcionais de um único problema. A figura 1 representa essa inversão:

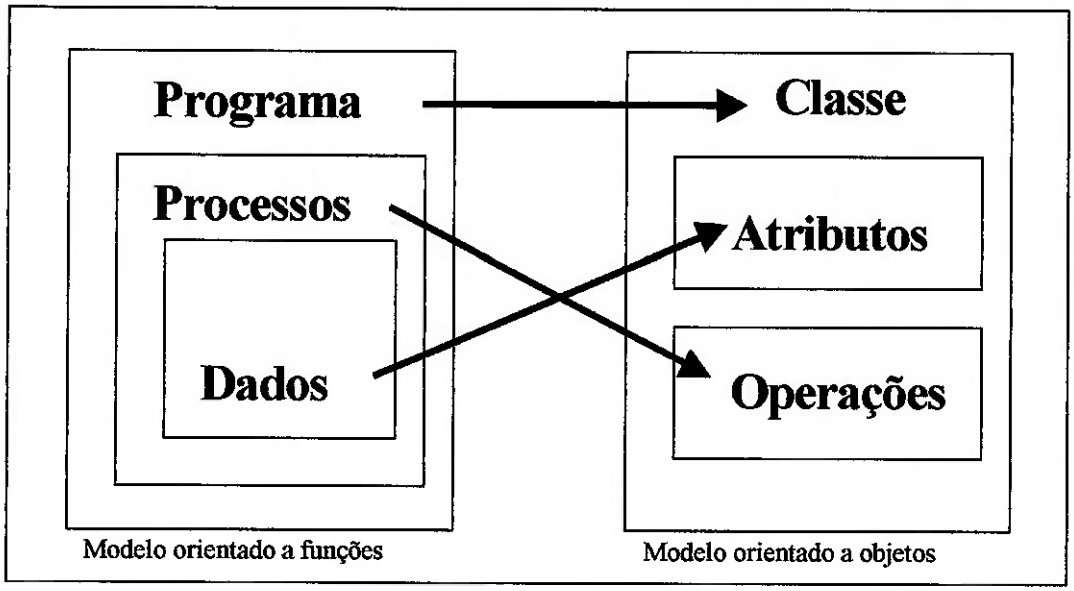


Figura 1 - Mudança de Paradigma

2.1.2 Robustez

A robustez em *software* é a habilidade de fácil recuperação em caso de falhas. Muitas linguagens e ambientes modernos orientados a objeto suportam a detecção e o manuseio de exceções.

2.1.3 Reutilização

Conforme muito bem colocado por Sintes (2002,p.15) : "Um construtor não inventa um novo tipo de tijolo cada vez que constrói uma casa. Um engenheiro eletricista não inventa um novo tipo de resistor cada vez que projeta um circuito. Então, por que os programadores continuam 'reinventando a roda' ? Uma vez que um problema esteja resolvido, você deve reutilizar a solução."

A tecnologia OO oferece uma modularidade de seus componentes para que possam ser integrados a diversos sistemas. Tais objetos podem ser combinados ou utilizados separadamente, pois apresentam baixa dependência externa e alta coesão interna. Assim sendo, é possível reutilizar classes orientadas a objetos bem feitas. Entretanto, a OO não garante código genérico. Criar classes bem feitas é uma tarefa difícil que exige concentração e atenção à abstração.

2.1.4 Confiabilidade

A reutilização permite que se escreva menos código, e quanto menos código houver, menor a probabilidade de erros. A estrutura da OO facilita os testes de unidade para cada classe criada, permitindo o teste durante o desenvolvimento, e não apenas no final.

2.1.5 Extensibilidade

Devido a sua característica não estática, o *software* deve sofrer constantes modificações e crescer com o passar do tempo, para permanecer útil. A técnica de Orientação a Objetos propicia vários recursos para estender código tal como: herança, polimorfismo, sobreposição, delegação e uma variedade de padrões de projeto.

2.1.6 Distributividade

Essa é uma característica que foi possível graças ao CORBA, uma arquitetura para dar suporte a sistemas orientados a objeto distribuídos por múltiplas plataformas. O CORBA permite que os objetos se comuniquem por meio de máquinas de diferentes modelos, executando sistemas operacionais diferentes e conectados por uma variedade de redes distintas.

2.1.7 Armazenabilidade

Um ODBMS manterá objetos e proporcionará o encapsulamento, a herança, o polimorfismo e outras características importantes orientadas a objeto.

2.1.8 UML

A UML é uma linguagem-padrão para a elaboração da estrutura de projetos de *software*. Contudo, não é uma linguagem de programação, e sim uma linguagem para modelagem. Ela pode ser aplicada à modelagem de vários tipos de sistemas: sistemas corporativos, baseados em WEB e até sistemas complexos embutidos de tempo real.

Devido a sua expressividade, ela abrange todas as visões necessárias ao desenvolvimento e implantação de sistemas, disponibilizando modelos para uso em vários níveis de abstração: conceitual, lógico e físico, assim como, para uso em várias perspectivas: estática e dinâmica. Por ser apenas uma linguagem, a UML é somente uma parte de um método para desenvolvimento de software, e é independente do processo.

A UML facilita muito a comunicação entre os profissionais de desenvolvimento de software. Por trás de cada símbolo empregado, existe uma semântica bem-definida. Um desenvolvedor poderá usar a UML para descrever seu modelo, e qualquer outro desenvolvedor, ou até outra ferramenta, será capaz de

interpretá-lo sem ambigüidades. Assim sendo, para explicação de vários conceitos de OO, usaremos representações em UML. Na referência bibliográfica, há literaturas sobre UML que podem ser consultadas.

2.1.9 Abstração

Entende-se por abstração o processo de simplificar um problema difícil.

Deve-se simplificar o mundo real e incluir apenas as partes que realmente afetam a simulação do problema. A figura 2, abaixo, apresenta uma situação em que somente a característica velocidade e o comportamento andar são importantes para o problema em análise. Entretanto, o carro é muito mais complexo, apresentando várias outras características e comportamentos, que pela abstração passam a ser desconsiderados.

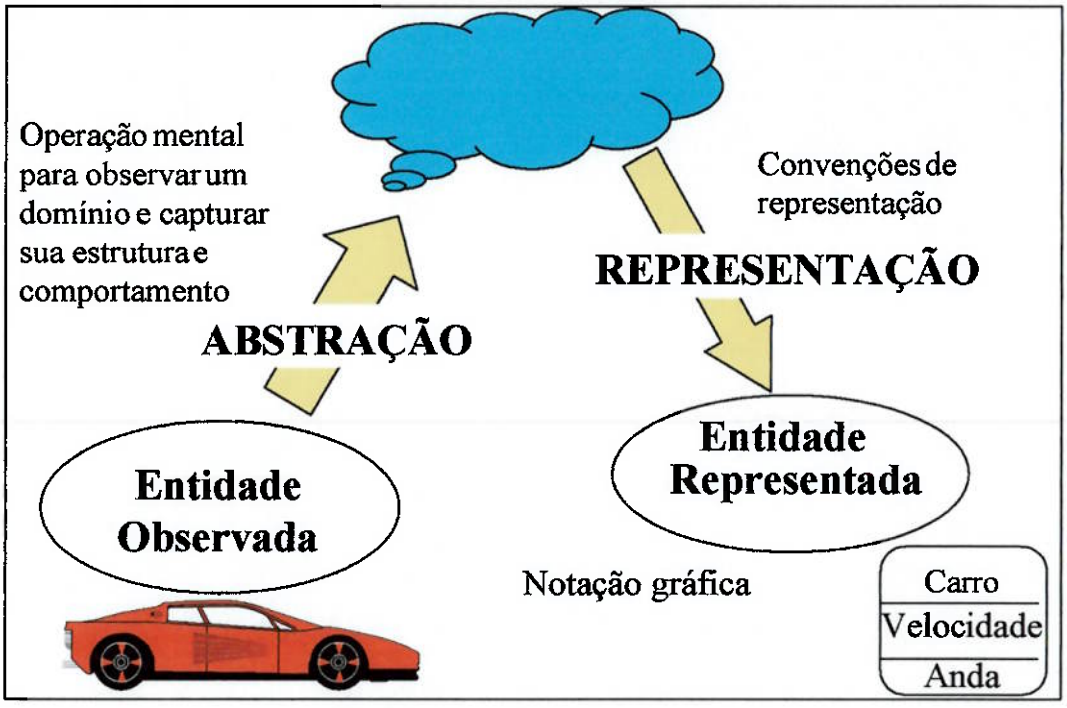


Figura 2 - Atividade de abstração

2.1.10 OBJETO

O conceito mais importante da Orientação a Objetos, é o próprio objeto. Ao observar o mundo à nossa volta, nos deparamos com uma imensa variedade de objetos que possuem propriedades e comportamentos as que afetam.

Assim como no mundo real, **tudo** na Orientação a Objetos é um objeto. A figura 3, abaixo, apresenta exemplo de objetos no mundo real.

- **Uma nota fiscal**
- **Uma empresa**
- **Uma tela com a qual o usuário interage**
- **Um desenho de engenheiro**
- **Uma aeronave**
- **Um vôo de avião**
- **Uma reserva de uma peça de teatro**
- **Um processo de atendimento de pedidos**
- **Um computador**

Figura 3 - Exemplos de objetos

Desta forma, Furlan (1998,p.17) sugere o seguinte conceito de objeto: "compreende-se por objeto a abstração de alguma coisa em um domínio de problema, de maneira autocontida e identificável, refletindo a capacidade de manter um estado de si, encapsulando valores de atributos e serviços exclusivos. Ele, todavia, não muda de identidade, comportamento ou atributos definidos por sua classe".

2.1.11 CLASSE

A seguir, são apresentados, a definição de classe e os critérios para a sua identificação.

Intuitivamente, agrupamos objetos semelhantes, chamamos de planta tanto um limoeiro quanto uma bananeira; porém, não o fazemos com uma pedra.

Esse processo de **classificação** é inerente ao ser humano. E a classe em *software* representa exatamente isso, pois define o conjunto de objetos que têm a mesma estrutura e o mesmo comportamento.

Uma classe é construída com os conceitos de abstração e encapsulamento.

O conceito de classe será detalhadamente explorado, pois conforme será abordado adiante, o Modelo de Classes é a base para o desenvolvimento de um sistema. A fim de solidificar esse conceito tão importante em OO, estão elencadas abaixo, definições de classe de diversos autores.

Martin (1994,p.219) afirma que "cada classe se parece com uma atriz, a quem se diz o que fazer e a quem se atribui certas responsabilidades. Ela pode desempenhar algumas responsabilidades por si própria, e realizar outras através de colaboradores, enviando-lhes, para tanto, solicitações.

O projetista é como um diretor, que deve designar os papéis a diferentes atores: ele lhes atribui uma classe, um nome e descrição de suas responsabilidades. O diretor coloca os atores em cena e aperfeiçoa o desempenho deles.".

"Uma classe é o estêncil do qual são criados objetos. Cada objeto tem a mesma estrutura e comportamento da classe na qual ele teve origem." (Meilir Page-Jones)

"As classes são como as formas com as quais se fazem biscoitos. Os objetos são os biscoitos. A 'massa do biscoito' em forma de memória terá de ser providenciada." (Horstmann e Cornell)

"Classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, métodos, relacionamentos e semântica." (BJR97)

"A biologia classifica todos os gatos, cães, elefantes e seres humanos como mamíferos. Características compartilhadas dão a essas criaturas separadas um senso de comunidade. No mundo do software, as classes agrupam objetos relacionados da mesma maneira." (Sintes)

A seguir, uma definição bem prática apresentada por Mello; Chiara e Villela (2002,p.15): "Enquanto uma classe é somente a codificação na forma de um arquivo texto, um objeto é uma instância de uma classe, ou seja, é uma porção de memória reservada para armazenar os dados e os métodos declarados na classe. Enfim, um objeto é a instanciação de uma classe na memória. A classe é o código-fonte escrito em um arquivo texto, enquanto o objeto é uma parte de um aplicação durante o processo de execução. No objeto, pode-se executar os métodos que foram definidos, além de acessar e alterar dados.".

A última definição acima diz respeito, especificamente, a classe de implementação, pois o termo classe é tratado mais amplamente logo no início do desenvolvimento de um sistema, abrangendo especificações que precedem a implementação, ou seja, as classes de negócio do domínio de um problema.

A figura 4, a seguir, mostra a relação entre classe e objeto. Há cadeiras com diferentes características: giratória, de ferro, de madeira, ou seja, são os diferentes objetos de uma mesma classe denominada cadeira.

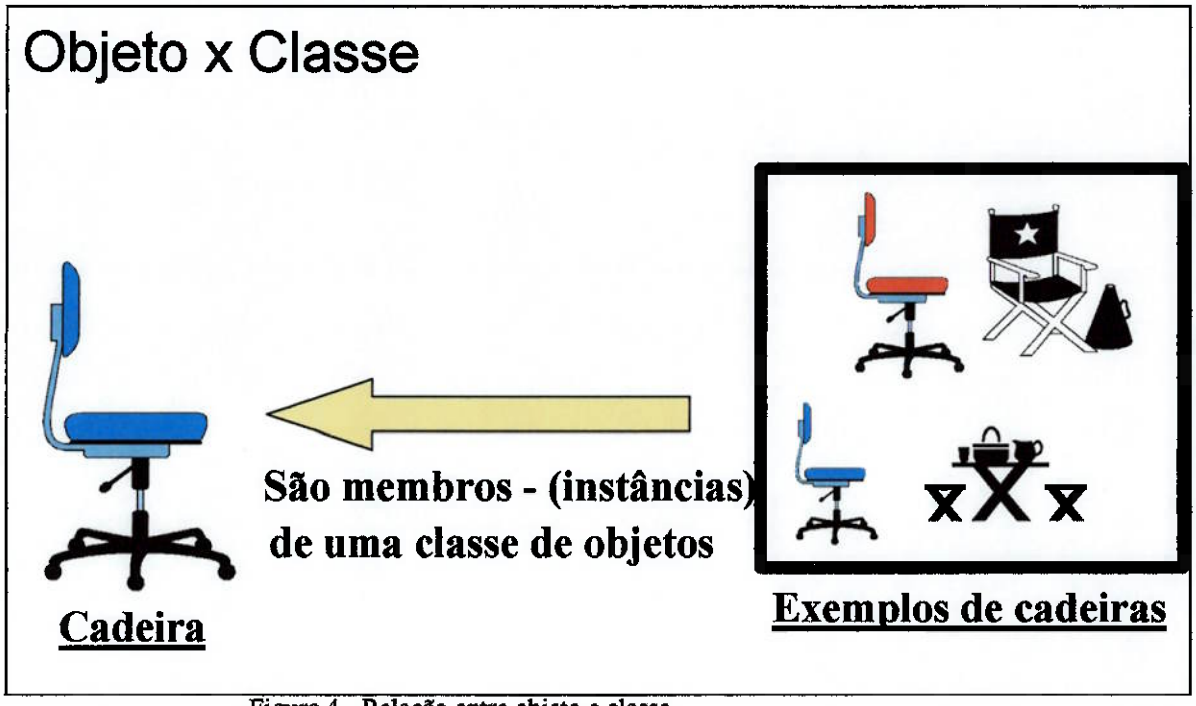


Figura 4 - Relação entre objeto e classe

Pode-se concluir, então, que ao criar uma classe, o objetivo é reunir dados e funcionalidades que estejam relacionados entre si. Essas funcionalidades devem ser determinadas segundo os objetivos da classe que tenham importância para o domínio do problema. Por exemplo, se estivermos determinando os tipos de dados da classe cliente para um sistema de cadastro de clientes de uma locadora de filmes, informações como nome e endereço do cliente são significativas; porém informações como peso e altura já não seriam. Exemplos de funcionalidades que atuam sobre tais dados podem ser alterar endereço do cliente, ou tirar cliente da lista de entrega de filmes em atraso.

É importante salientar que uma classe não deve ter muitas responsabilidades. Se ela for muito complexa, poderá ser subdividida, pois as unidades menores têm mais possibilidades de reutilização do que as maiores e mais complicadas.

Determinar as classes é estabelecer um modelo conceitual para um sistema que se deseja desenvolver. Identificar esses conceitos é a alma da análise orientada a objetos, sendo um esforço que vale a pena devido ao retorno que traz na fase de projeto e implementação.

"O passo essencialmente orientado a objetos na análise ou na investigação é a decomposição do problema em conceitos" (Craig Larman)

Para resolver problemas complexos utilizamos a decomposição, uma estratégia comum para lidar com a complexidade, pela divisão do problema em unidades compreensíveis. Na análise estruturada, o critério usado para essa decomposição são os processos, já na OO, a dimensão usada é fundamentalmente por conceitos (classes).

Todavia, como identificar as classes?

Segundo sugere Larman (2000,p.105), pode-se fazer uma lista de categorias de conceitos que pode ser seguida para que as classes sejam determinadas. A tabela I apresentada abaixo, fornece exemplo de uma lista de conceitos candidatos a determinarem as classes de um domínio de problema. Essa tabela foi extraída do livro "Utilizando UML e Padrões" referente a um exemplo de sistema de venda de passagens aéreas.

Categoria de Conceito	**Exemplos de classes**
Objetos físicos ou tangíveis	Terminal de Ponto-de-venda Aeronave
Especificações de projetos ou descrições de coisas	Especificação de Produto Descrição de Vôo
Lugares	Loja Aeroporto
Transações	Venda Pagamento

	Reserva
Linhas de itens de transações	Linha de Item de Venda
Papéis desempenhados por pessoas	Caixa Piloto
Contêineres de outras coisas	Depósito Armário
Outros sistemas de computador ou dispositivos eletromecânicos externos ao nosso sistema	Sistema de Autorização de Cartão de Crédito Controle do Tráfego Aéreo
Conceitos de substantivos abstratos	Fome Aerofobia
Organizações	Departamento de Vendas Linha Aérea
Eventos	Venda Reunião Vôo
Regras e estratégias	Política de Reembolsos Política de Cancelamentos
Catálogos	Catálogo de Produtos Catálogo de Peças
Registros financeiros, trabalhistas, de contratos, etc	Recibo Diário de Caixa Contrato de Emprego
Serviços e instrumentos financeiros	Linha de Crédito Ação

Tabela I - Subsídios para construir um modelo conceitual de classes

É bastante interessante essa técnica proposta por Larman para identificar classes em um domínio de negócio, pela apresentação de todas as possibilidades em que se deve pensar para definir uma classe.

Outra estratégia proposta por Abbot apud Larman bem simples é identificar os substantivos nas descrições de um domínio de problema e considerá-los como candidatos a classes ou atributos de classes a serem incluídos no modelo

conceitual. Entretanto, deve-se tomar cuidado com essa técnica para não fazer um mapeamento puramente mecânico de substantivos para classes, pois diferentes substantivos podem representar o mesmo conceito ou atributo.

2.1.12 Operação

Compreende-se por operação o comportamento dos objetos. No mundo real, as operações manipulam as características dos objetos. Por exemplo, uma bicicleta corre mais rápido porque a corrente é influenciada, sendo esta uma característica da bicicleta.

Na Orientação a Objetos, operações são usadas para ler ou manipular os atributos de um objeto. As operações em uma classe se referem apenas às estruturas de dados daquele tipo de objeto. Elas não devem acessar diretamente as estruturas de dados de outro objeto, caso desejem fazê-lo, as operações devem enviar uma mensagem àquele objeto. A figura 5 abaixo, apresenta exemplo de operações de uma classe.

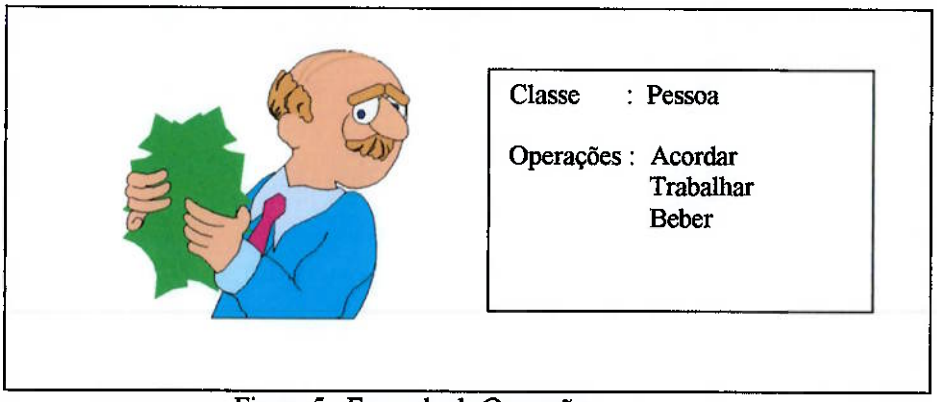


Figura 5 - Exemplo de Operações

2.1.13 Atributo

Considera-se atributo tudo o que objeto "conhece" sobre si mesmo, e é definido na classe à qual pertence. Esses são os tipos de dados que uma classe possui; são equivalentes ao atributos que especificamos em entidades do Modelo Entidade-Relacionamento. A seguir, na figura 6, um exemplo de atributos da classe pessoa.

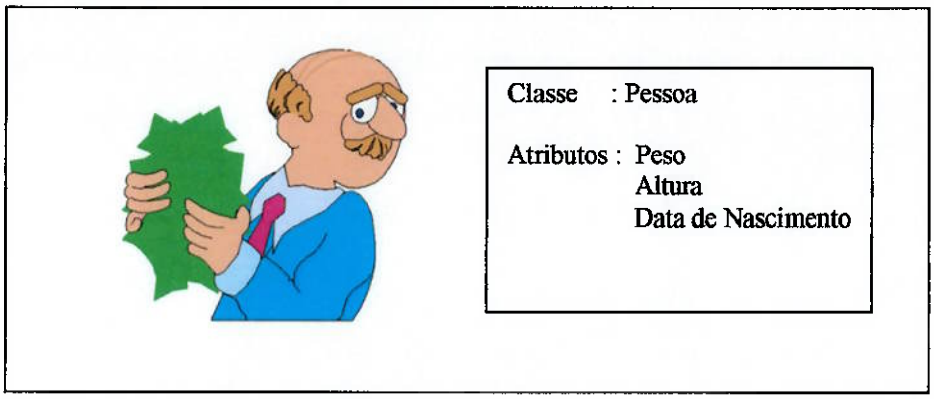


Figura 6 - Exemplo de atributos

2.1.14 Método

Tudo que o objeto "pode fazer" está expresso em seus métodos, os quais são as funcionalidades que uma classe possui.

Os métodos especificam a maneira como as operações são codificadas no *software*, constituindo a implementação da Operação. Desta forma, eles nada mais são do que procedimentos e funções das linguagens tradicionais.

Abaixo, pode-se constatar os tipos de métodos:

Método Construtor

Um construtor tem o objetivo de alocar a memória requerida para um novo objeto, ou seja, instancia o objeto.

Método Destruidor

É exatamente o contrário do construtor, pois desaloca a memória utilizada pelo objeto, destruindo-a efetivamente.

Método de Instância

Os métodos de instância são aplicáveis a objetos individuais, modificando e acessando os atributos de instâncias. Por exemplo, as contas-correntes, o método depositar e sacar são aplicáveis a contas individuais, portanto são métodos de instância.

Método de Classe

Os métodos de classe são aplicáveis somente a classe. Por exemplo, um banco precisa mudar com freqüência o valor do saldo mínimo devido às pressões do mercado financeiro, portanto, ele precisa de um método que atualize o valor do saldo mínimo. Por dizer respeito a uma classe como um todo, denomina-se este como um método de classe.

A seguir, na figura 7, há um exemplo prático de um classe implementada em Java:

```
// declaração inicial da classe cliente

public class cliente

{
   // atributos da classe cliente

   String nome;
   String endereco;

  // métodos da classe cliente
  public void atualiza_nome(String novo_nome)
  {
    nome = novo_nome;
  }

  public String obtem_nome()
  {
    return nome;
  }
} // fim da declaração da classe Cliente
```

Figura 7 - Exemplo de métodos em Java

2.1.15 Relacionamento

Os relacionamentos ligam classes/objetos entre si criando relações lógicas entre essas entidades. Esses relacionamentos são similares aos relacionamentos estabelecidos nos DER's.

Em OO, há 3 tipos de relacionamentos: associação, generalização e dependência, as quais serão abordadas, a seguir:

2.1.16 Associação

É uma relação que descreve um conjunto de vínculos entre elementos de modelo (Furlan,1998), assim como representa a ligação entre classes ou objetos, podendo ambas conhecerem-se.

2.1.17 Generalização

É o relacionamento que define a herança entre classes.

2.1.18 Dependência

É um relacionamento entre uma classe independente e outra dependente. Uma modificação em um objeto da classe independente afetará diretamente objetos da classe dependente.

2.1.19 Encapsulamento

Os componentes eletrônicos e os seus dados não podem ser acessados diretamente. (Está escrito: CUIDADO – NÃO ABRA – RISCO DE CHOQUE ELÉTRICO). Você só pode fazer uso das operações especificadas. O encapsulamento evita interferência no interior e também esconde a complexidade dos componentes.

Cada objeto tem, encapsuladas, a estrutura dos dados e as operações. Uma estrutura de dados localiza-se no núcleo de um objeto, o qual é manipulado pelas operações que implementam as operações permitidas. A estrutura de dados pode ser utilizada apenas com esses métodos. Esta restrição de acesso é chamada de Encapsulamento, que protege os dados contra corrupção.

Abaixo na figura 8, o esquema de caixa preta que bem representa o encapsulamento :

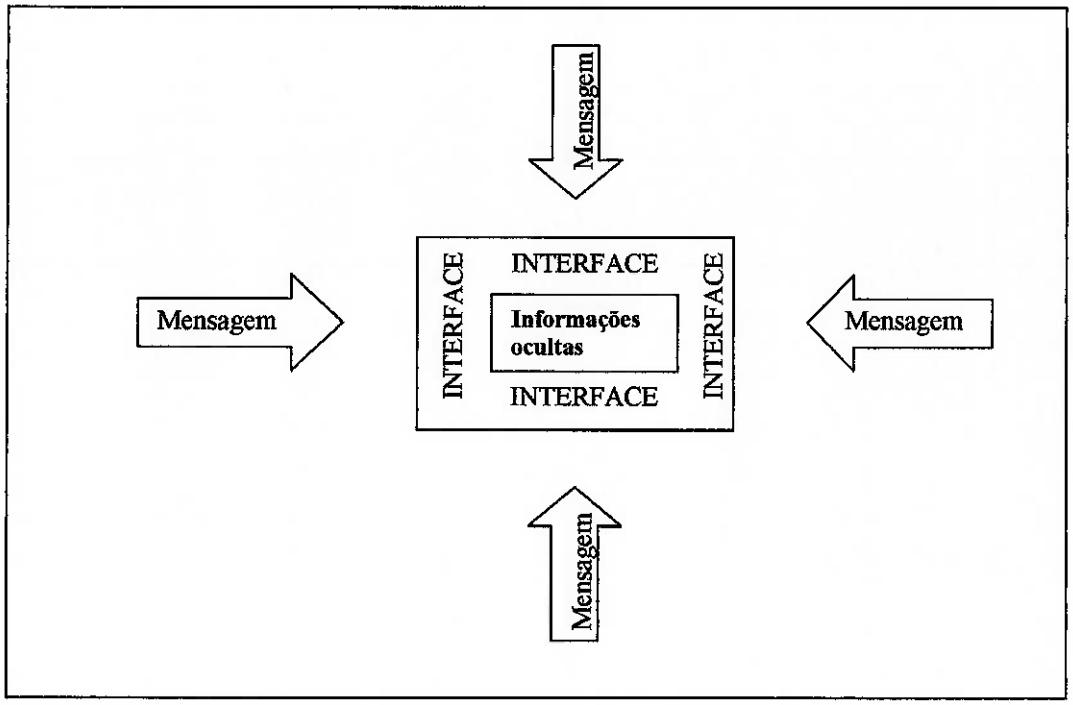


Figura 8 - Representação do encapsulamento

O encapsulamento tem uma finalidade similar à da tradicional sub-rotina. Entretanto, o encapsulamento é estruturalmente mais sofisticado. Uma simples sub-rotina não apresenta os recursos que o encapsulamento da OO tem:

- Determinar que atributos de um objeto sejam acessados e atualizados somente pelas operações da próprio objeto, pois tem-se o recurso da determinação da visibilidade dos atributos; basta defini-los como "*private*" e mais nenhum outro objeto pode acessar um atributo tomando diretamente a(s) variável(eis) subjacente(s) que implementa(m) o atributo.

- Possibilitar que o estado de um objeto seja acessível ou modificável somente pela *interface* provida pelo encapsulamento.

Observa-se assim que, o Encapsulamento permite criar objetos bem definidos e independentes.

Um exemplo prático, se for necessário utilizar um objeto que valide o CNPJ, não importa como essa validação seja implementada, interessa apenas que funcione para ser reutilizada. Esse segredo da implementação é ocultado pelo encapsulamento. É uma simples comparação com o mundo real, todos os objetos que compramos: *DVD-player*, TV, secador de cabelos, computador, enfim, não interessa o mecanismo interno de funcionamento e sim que a função desejada seja executada com sucesso.

2.1.20 Qualificador

Denomina-se qualificador um dos recursos da OO que implementa o Encapsulamento, através do qual se define a visibilidade de atributos ou métodos. Na tabela II, abaixo, pode-se observar exemplos de qualificadores:

Qualificadores	Grau de visibilidade
Private	O método ou atributo pode ser acessado somente dentro da própria classe
Public	O método ou atributo pode ser acessado externamente por outro código
Protected	O método ou atributo pode ser acessado pela própria classe ou por classes-filhas (aquelas que herdam desta classe)
Package	O método ou atributo pode ser acessado pela própria classe ou por classes que participem do mesmo pacote

Tabela II - Exemplo de Qualificadores em Java

2.1.21 Interface

Pelo encapsulamento, um objeto requisitante de um serviço pode não saber como um objeto requisitado executa suas operações, mas precisa, no mínimo,

saber o que o objeto sabe fazer ou que informação ele conhece. Para tal atividade existe a *interface* que define os serviços que um objeto pode realizar e as mensagens que ele recebe.

Esse conceito é muito importante em OO, pois os objetos são conhecidos pelas suas *interfaces*. Então, não há como saber algo ou solicitar um serviço para o objeto sem ter conhecimento de sua interface. A interface de um objeto nada diz sobre sua implementação, pois diferentes objetos estão livres para implementar as solicitações de diferentes maneiras. Isso significa que dois objetos que tenham implementações completamente diferentes podem ter interfaces idênticas.

Uma interface define um protocolo de comportamento que pode ser implementado por qualquer classe de uma hierarquia. Quando uma classe se propõe a implementar uma interface, ela deve definir o conteúdo de todos os métodos declarados na interface, a qual pode ter várias formas de implementação.

Podemos, também, entender a Interface como um mecanismo de "proteção" do objeto, como colocado por Arídio Silva, em que a Interface é a capa protetora do objeto, "... o interface assegura que todas as interações com o objeto aconteçam por intermédio de um sistema pré-definido de mensagens que o objeto garante entender e controlar corretamente." , "... os dados dentro do objeto são acessados somente pelos métodos que implementam o interface do objeto.", "... os objetos não tocam nas estruturas de dados de outros objetos.".

O uso de interfaces também é interessante pelo poder de flexibilidade que elas oferecem ao sistema por meio do protocolo que elas criam.

Um exemplo de como isso funciona:

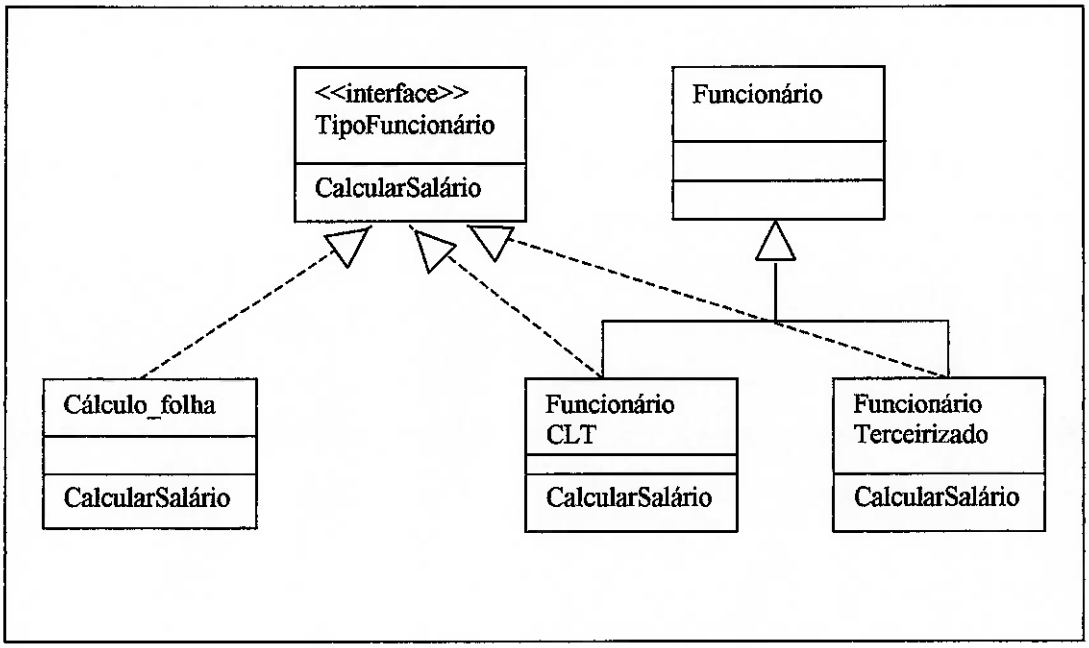


Figura 9 - Utilização de interfaces

A figura 9 apresenta uma classe Calculo_Folha que utiliza a interface TipoFuncionário como protocolo de comportamento. Segundo a definição da interface TipoFuncionário, existe um método chamado CalcularSalário que calcula uma série de operações de acordo com o tipo de funcionário.

Na mesma figura também é apresentada a hierarquia de classes na qual a superclasse Funcionário possui as subclasses Funcionário_CLT e FuncionárioTerceirizado, que implementam a interface TipoFuncionário. No entanto, cada uma das subclasses apresenta uma implementação diferente para a interface TipoFuncionário.

Na classe Funcionário_CLT, a implementação de CalcularSalário executa um cálculo considerando vários impostos FGTS, INSS, etc, ao contrário da classe FuncionárioTerceirizado que calcula somente as horas trabalhadas calculando apenas o IR sobre a fonte.

Quando se deseja executar um cálculo de salário, faz-se uma chamada para o método CalcularSalário na classe CálculoFolha, na qual é passada como parâmetro uma classe que implementa a interface TipoFuncionário. O método CalcularSalário da subclasse do parâmetro passado é que será executado. Esse exemplo de aplicação de interfaces garante maior flexibilidade à classe Cálculo_Folha (que recebe um parâmetro do tipo TipoFuncionário e sabe que na classe que implementa essa interface existe um método chamado CalcularSalário implementado).

2.1.22 Estado

Um estado é uma situação na vida de uma objeto durante a qual o objeto satisfaz alguma condição, realiza alguma atividade ou aguarda um evento [Booch]. Normalmente, o estado é determinado pelos valores dos atributos e ligações com outros objetos; um objeto muda de estado quando um evento ocorre.

2.1.23 Retenção de Estado

A retenção de Estado é outra vantagem da OO sobre as técnicas tradicionais. Quando um módulo de rotinas tradicionais termina de ser executado e retorna a quem o chamou, ele "morre"; se executado novamente, todas as variáveis são inicializadas, não podendo ter mais acesso às informações da transação anterior.

Contudo, isso já não ocorre com um objeto, pois ele retém informações dentro dele durante um período indefinido de tempo, ou seja, um objeto não "morre" quando ele termina de ser executado; portanto um objeto retém estado.

2.1.24 Comportamento

O comportamento é o meio pelo qual o objeto passa de um estado para o outro. Normalmente, isso se dá mediante uma condição/ação.

2.1.25 Identidade de Objeto

Tal como no mundo real, em que todos os entes são únicos, os objetos possuem sua identidade e são distinguíveis.

Gêmeos idênticos são duas pessoas distintas, como também todos os carros idênticos que saem de uma linha de produção automobilística são distintos. Da mesma forma, os objetos com os mesmos valores de atributos também são distintos.

2.1.26 Mensagens

Assim como nós não vivemos isolados no nosso mundo, os objetos também não vivem isolados em seu mundo, e se o fizessem, não teriam utilidade alguma.

Outra analogia com o mundo real é o princípio básico da física: o da "Ação e Reação", pois essa lei também se aplica a objetos. Considera-se, dessa forma que, deve haver um estímulo enviado a um objeto, ou seja, a ação para que uma das operações desse objeto seja executada, a reação. Esse estímulo é a mensagem.

Os objetos interagem e se comunicam uns com os outros, através do envio de mensagens que requisitam aos objetos a execução dos seus métodos.

As mensagens nada mais são do que as chamadas de funções e procedimentos das linguagens tradicionais, podendo, também, enviar parâmetros de Entrada e/ou Saída consigo, que podem ser compostos de outros objetos.

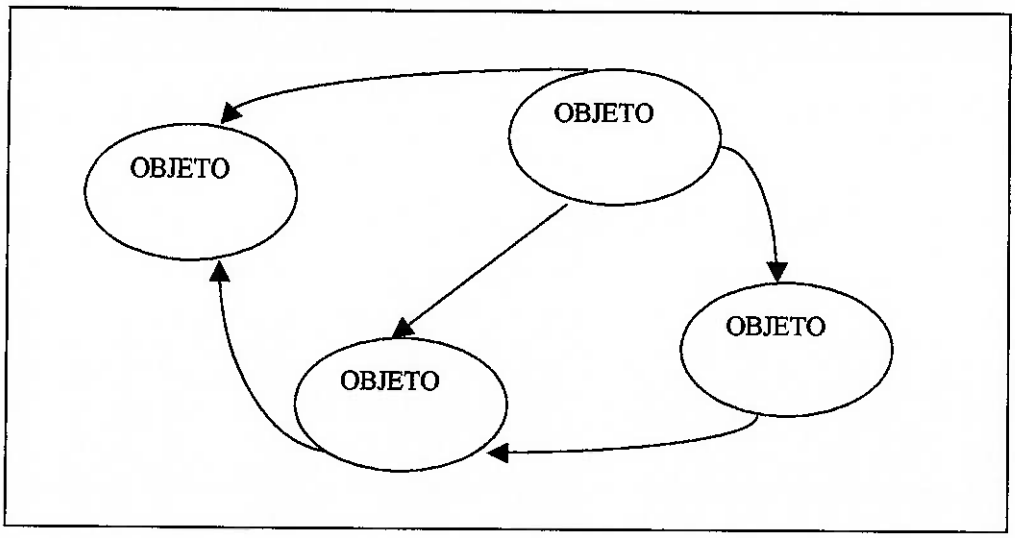


Figura 10 - Envio de mensagem entre objetos

2.1.27 Herança

Traçando um outro paralelo da OO com a nossa vida, sabemos que todos os seres vivos do planeta herdam os padrões dos genes de seus ancestrais. Isso faz com que tenhamos semelhanças em estrutura e comportamento. Esse mecanismo também ocorre com os objetos. Da mesma forma, uma **subclasse** (aquela que herda as características de determinada classe) inclui os dados e comportamentos, ou seja, atributos e métodos, da **superclasse** (aquela de quem se herda as características).

O interessante da herança é poder acrescentar atributos e métodos que são exclusivos da subclasse. Além disso, permite também redefinir qualquer comportamento herdado.

Ao processo de definição de subclasses, em que uma classe herda elementos de outra e assim sucessivamente, dá-se o nome de hierarquia de classes. Cada nível abaixo da superclasse do sistema acrescenta funções específicas.

A figura 11 a seguir, apresenta uma hierarquia de classes em que Funcionário é a superclasse, Funcionário CLT e Funcionário Terceirizado são subclasses. No nível abaixo da superclasse, existem especializações da superclasse,

ou seja, subclasses que herdam as características da classe Funcionário e nas quais podem ser adicionadas características específicas.

Uma boa forma de identificar um herança é fazer a pergunta "é uma" para a subclasse em relação a superclasse. Por exemplo: um funcionário CLT é um funcionário ? A resposta é sim. Caso a resposta seja não, ou a herança não seja válida ou durante a fase de projeto foi necessária a criação de tal estrutura.

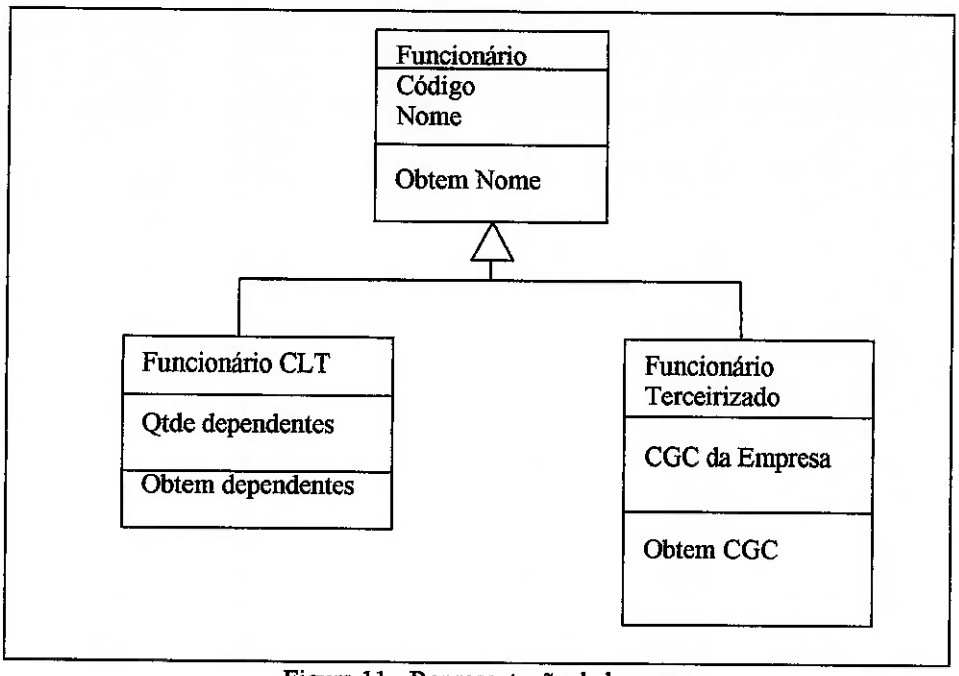


Figura 11 - Representação de herança

É a herança que permite que se construa *software* incrementalmente pois possibilita a reutilização.

A herança resulta em código confiável, pois adicionamos apenas o código que descreve a diferença entre a superclasse e a subclasse, portanto, cada classe pode ter código menor. Cada classe pode ser altamente especializada no que faz, assim menos código significa menos erros.

Novas funcionalidades podem ser incluídas com mínima ou nenhuma alteração do código já existente. Isso seria muito difícil pelas técnicas tradicionais,

sendo que na OO essa característica já está incorporada na linguagem, basta saber utilizar corretamente.

2.1.28 Generalização x Especialização

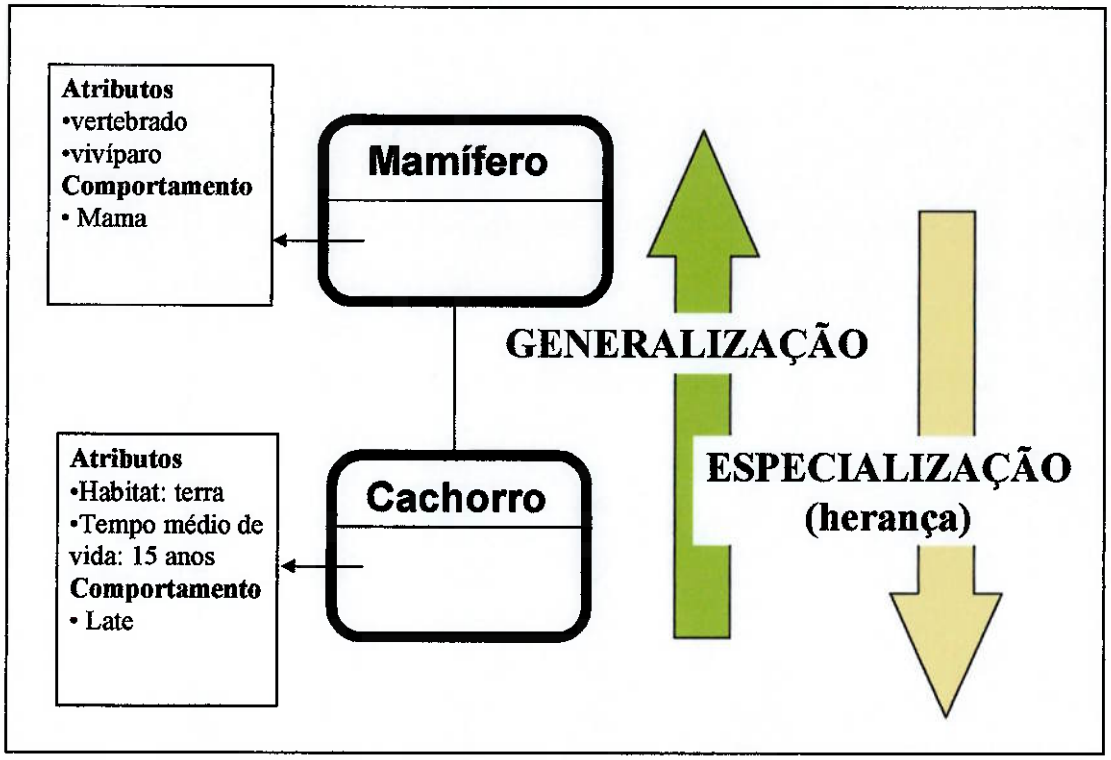


Figura 12 - Generalização x Especialização

A figura 12 apresenta os relacionamentos que existem na herança, um relacionamento entre o geral e o específico. Se existe um relacionamento de generalização, então sabe-se que é possível substituir uma classe filha pela classe progenitora.

2.1.29 Polimorfismo

Na vida real, as mesmas operações são realizadas por diferentes objetos o tempo todo. Abre-se uma porta, abre-se uma janela e assim por diante.

Há vários exemplos de Polimorfismo ocorrendo no mundo real; o exemplo de controle remoto que serve tanto para a tv como para o vídeo é um deles. A mesma operação ligar, aplicada a objetos distintos.

Até com a psicologia, Tony Sintes faz um paralelo ao afirmar que " ... o polimorfismo é o distúrbio das múltiplas personalidades do mundo do software, pois um único nome pode expressar muitos comportamentos diferentes".

A aplicação do conceito de polimorfismo em linguagens orientadas a objeto segue esses mesmos princípios. Em software, diferentes objetos podem responder a uma mesma mensagem genérica, mas cada um deles podem interpretar a mensagem de uma forma diferente e peculiar a si mesmo.

Como o próprio nome já diz, Polimorfismo significa muitas formas. Permite que um único nome de classe ou nome de método represente um código diferente, selecionado por algum mecanismo automático.

Segundo Sintes (2002,p.126) há 4 tipos de Polimorfismo:

1. Polimorfismo de inclusão

O polimorfismo de inclusão permite que um objeto expresse muitos comportamentos diferentes, em tempo de execução.

2. Polimorfismo paramétrico

Do mesmo modo que o polimorfismo de inclusão, o paramétrico permite que um objeto ou método opere com vários tipos de parâmetro diferentes.

3. Sobreposição

A sobreposição permite que você sobreponha um método e saiba que o polimorfismo garantirá que o método correto sempre será executado.

4. Sobrecarga

A sobrecarga permite que você declare o mesmo método várias vezes. Cada declaração difere simplesmente no número e no tipo de argumentos. A conversão faz um método parecer polimórfico, convertendo argumentos nos tipos de argumentos esperados pelo método.

O polimorfismo simplifica o código. Em vez de ter de programar casos especiais de cada tipo de objeto que poderia manipular, se escreve simplesmente um caso. Todavia, pelas técnicas tradicionais, teria de atualizar o código sempre que adicionasse uma nova subclasse.

O polimorfismo permite que se escreva código mais curto e mais inteligível, que também é mais flexível para os requisitos futuros.

A partir dos mecanismos da OO, principalmente, o tripé Encapsulamento, Herança e Polimorfismo, há maior potencial em arquitetar software que possua as qualidades mais desejáveis: natural, confiável, reutilizável, manutenível e extensível.

Com as técnicas tradicionais, poderíamos tentar projetar módulos reutilizáveis, herança e até polimorfismo. Entretanto, seria um trabalho imenso injustificável, tendo em vista que essas técnicas lhes são inerentes, e é melhor não reinventar a roda, pois a corrida por produtividade com qualidade é grande.

3 ABORDAGEM PARA COMPREENSÃO E APLICAÇÃO DA TÉCNICA

3.1 Migração da Técnica Estrutura para Orientação a Objetos

Considerando o intuito deste estudo de elaborar e implementar técnicas para profissionais interessados em migrar da análise estruturada para a OO, de acordo com os pressupostos apresentados no capítulo anterior, no presente capítulo, serão explicitadas e ilustradas, as técnicas e modelos utilizados para tal finalidade.

As figuras 13 e 14, a seguir ilustram as atividades a serem exercidas utilizando a Análise Estruturada e a Orientação a Objetos em cada etapa do desenvolvimento de sistemas. As atividades exercidas ao utilizar a técnica Orientada a Objeto serão descritas a partir da comparação com as técnicas de Análise Estrutura nas fases correspondentes, visando auxiliar o profissional que esteja nesse processo de transição, a fazê-lo sem muita dificuldade.

O paralelo entre as duas técnicas é feito para que o profissional da Análise Estruturada desmistifique a Orientação a Objetos, e não exagere ao classificá-la mais complexa do que ela realmente é, assim como possa fazer a migração rapidamente, pois a analogia é um dos métodos mais fáceis para o aprendizado, ao mostrar os aspectos semelhantes de novos conceitos com aqueles já conhecidos. Além disso, são apresentados exemplos simples para o fácil entendimento.

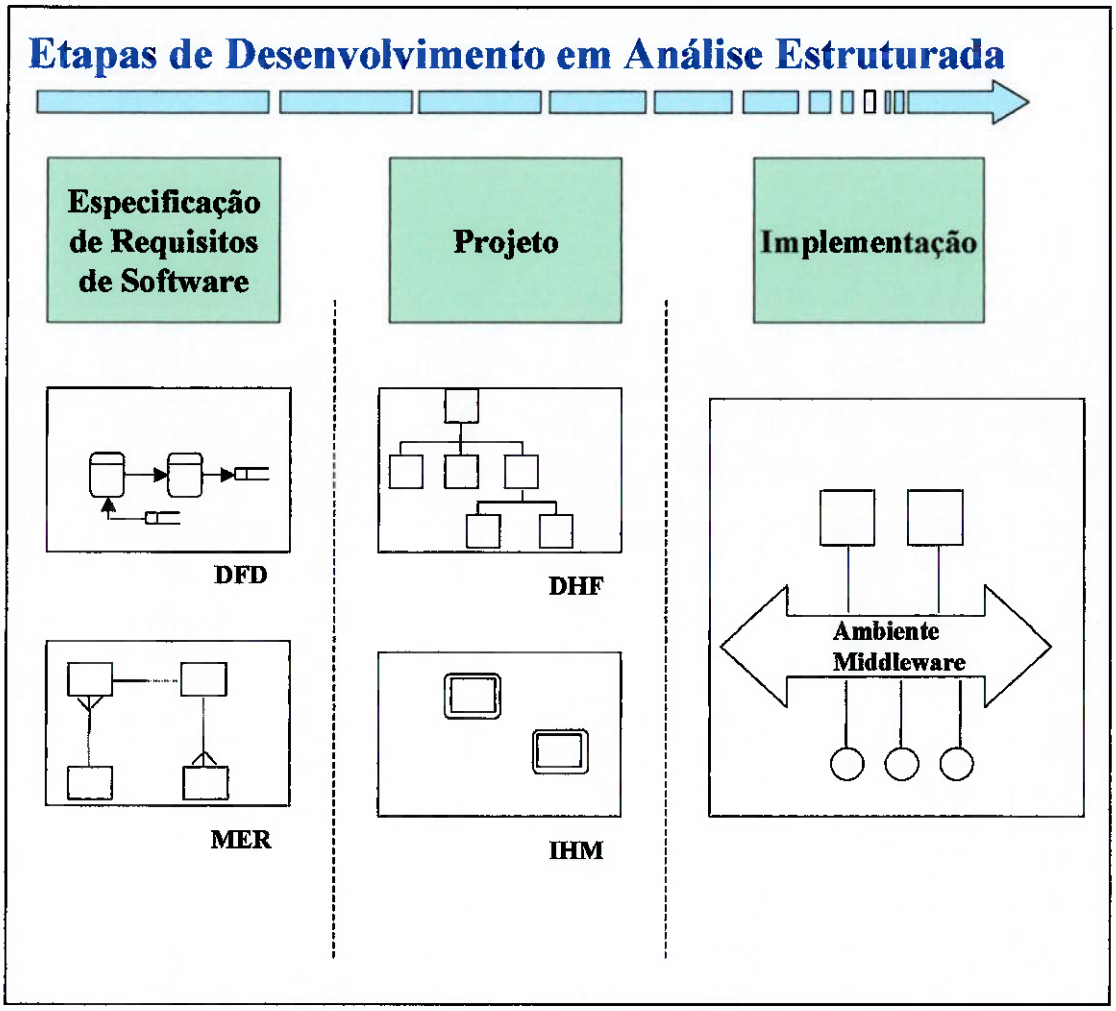


Figura 13 - Etapas de Desenvolvimento em Análise Estruturada

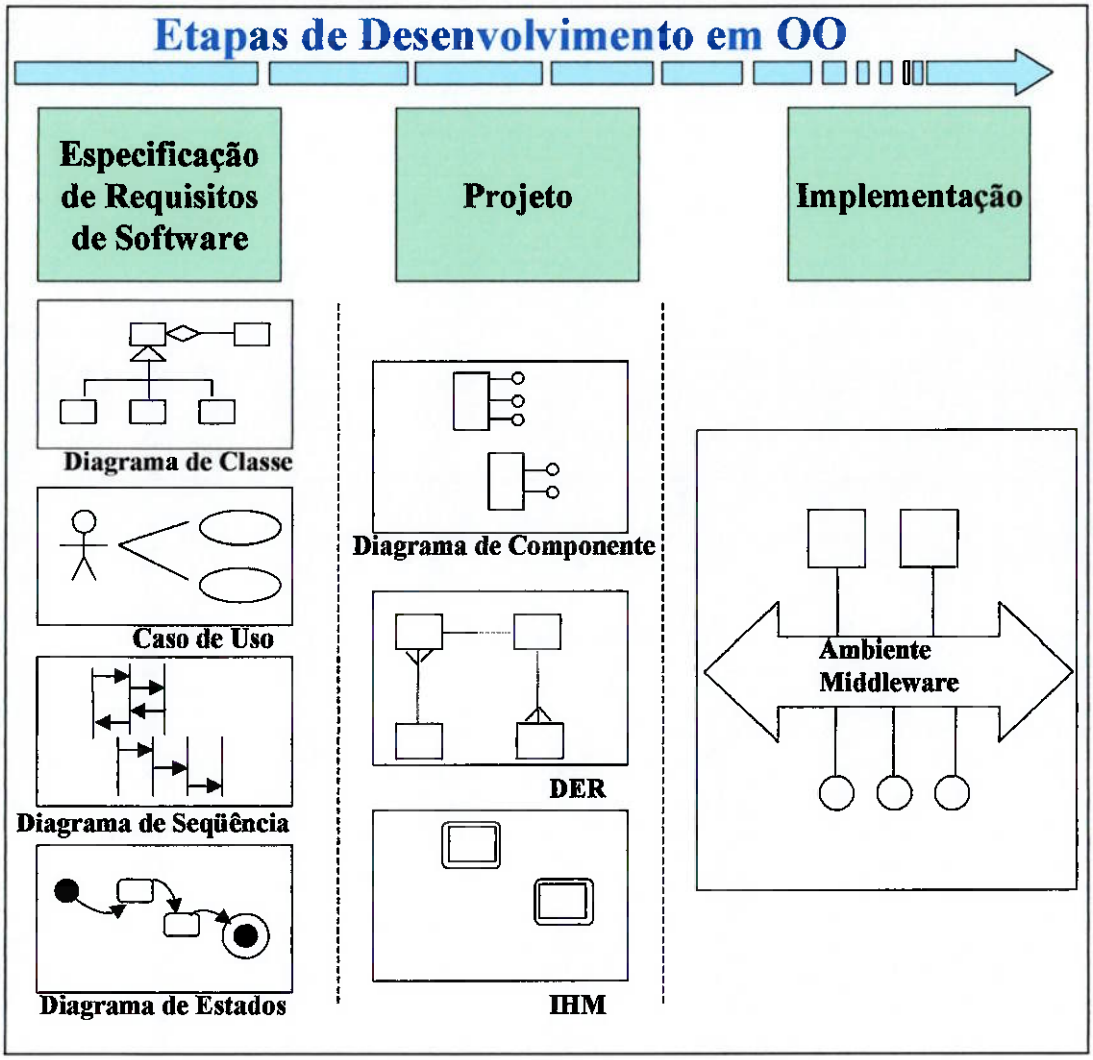


Figura 14 - Etapas de desenvolvimento em orientação a objetos

A seguir, são elencadas as atividades em cada fase do processo de desenvolvimento em OO. No final de cada atividade e fase da Orientação a Objetos, há uma conclusão para a comparação com os artefatos da fase correspondente da Análise Estruturada, além da análise do grau de dificuldade ao desempenhar as tarefas.

As observações sobre os artefatos da Análise Estruturada não se prestam a ensinar seus conceitos, é pertinente elucidar que visam reforçar características que serão contrapostas com os artefatos da Orientação a Objetos.

A tabela III mostra os artefatos produzidos em ambas as metodologias:

Fases de Desenvolvimento	Artefatos produzidos em Orientação a Objetos	Artefatos produzidos em Análise Estruturada
Fase 1 – Especificação dos Requisitos de Software	• Diagrama de Classes • Diagrama de Casos de Uso • Diagrama de Seqüência • Diagrama de Estados	• Diagrama de Fluxo de Dados • Diagrama de Entidade-Relacionamento
Fase 2 - Projeto	• Diagrama de Componentes • Diagrama de Entidade-Relacionamento • IHM	• Diagrama Hierárquico de Funções • IHM
Fase 3 - Implementação	• Codificação	• Codificação

Tabela III - Migrando da Análise Estruturada para a Orientação a Objetos

3.1.1 FASE 1: Especificação dos Requisitos de Software

A verificação do plano, cenário e regras de negócio, fluxo de trabalho de áreas envolvidas e a descrição de requisitos iniciais que o sistema deva contemplar, é comum a **ambas as técnicas**.

A seguir, serão descritos as atividades desempenhadas e os artefatos produzidos utilizando a metodologia Orientada a Objeto, durante a fase de Especificação dos Requisitos.

3.1.1.1 Desenvolvimento do Diagrama de Classes

Em OO, pode-se iniciar a modelagem de um sistema esboçando o Modelo de Classes que será a base para todo o desenvolvimento subseqüente. Já na análise estruturada iniciamos a modelagem com o DFD. Como esses dois diagramas sinalizam um ponto de partida, cada qual em sua técnica, serão feitas comparações importantes entre o DFD e o Diagrama de Classes. Para o melhor entendimento, encontram-se no anexo B, nas figuras 27 e 28, exemplos de um DFD e um Diagrama de Classes respectivamente, referentes a um mesmo domínio de problema. Este exemplo foi colocado para realçar as comparações entre os artefatos da Orientação a Objetos e da Análise Estruturada.

O exemplo utilizado para comparação entre o DFD e o Diagrama de Classes representa os processos envolvidos na atividade de importação de arquivos de extrato bancário de um sistema de Tesouraria. No DFD toda a inteligência do sistema está DISTRIBUÍDA através das funcionalidades. Contudo, no Modelo de Classes toda a inteligência do sistema está CONCENTRADA através das classes.

Quem ganha a cena no DFD são os processos, os depósitos de dados são atores coadjuvantes nessa peça que é o desenho do funcionamento do Sistema de Informações. Os depósitos de Dados são apenas suporte para os processos, fornecendo-lhes as informações necessárias para executar suas funções, pois os

processos comandam a elaboração do DFD. No âmbito do DFD, os processos são mais importantes que os dados. Na linha de raciocínio do profissional de Análise Estruturada, procura-se representar no DFD o fluxo de trabalho dos usuários.

Quem ganha a cena no Diagrama de Classes são os dados, porém não organizados em meros depósitos como no DFD, mas como entidades fortes e importantes para o negócio que o Sistema de Informações retrata.

Para desenvolver um esboço das classes, é necessário ter estudado, como na Análise Estruturada, tanto conceitos das regras de negócio quanto fluxos de trabalhos, mas não daremos ênfase aos processos como descrito na elaboração do DFD. Observam-se as entidades estabelecendo a relação estrutural entre elas.

No DFD, os depósitos de dados vão sendo definidos em função do que os processos precisam, para serem executados. Ao executar um processo, pergunta-se: que informações de entrada serão necessárias para fornecer ao processo? Que informações o processo irá gerar? E as diferentes categorias de informações geram depósitos distintos de dados. Os dados vão precisar "trafegar" de um processo para outro, por isso o nome Diagrama de Fluxo de Dados.

Observando o desenho de um DFD, parece que os processos são "Estáticos" e a parte "Dinâmica" são os dados, que ficam transitando de um processo a outro. Nessa ótica o DFD fica um pouco contraditório, pois os dados naturalmente são estáticos e mais estáveis e os processos são por natureza "dinâmicos", e essa é a interpretação correta que será verificada na Orientação a Objetos, já no Diagrama de Classes.

Assim, no Diagrama de Classes é preparada uma estrutura fortificada onde os processos deverão transitar, obtendo-se uma visão mais lógica e natural.

A inversão entre funções e estruturas de dados, citada anteriormente, no Capítulo de Fundamentos Conceituais, no item Mudança de Paradigma, poderá

ser observada. Desta forma, as classes tomaram o lugar dos processos em grau de importância nos respectivos diagramas, e o que aparece nas "caixinhas", que são as partes mais importantes focalizadas nos diagramas, são as classes.

Abaixo, está apresentada a análise da inversão importante da Orientação a Objetos em relação à Análise Estruturada:

- Entidades representadas como depósitos de dados no DFD da figura 27 do Anexo B, tais como Empresa, Banco, Agência, Conta-corrente, Estrutura EDI, Layout, Extrato, Saldo e Tradução Banco-Empresa são agora representadas como classes na figura 28 do Anexo B, pois são entidades fortes do domínio do problema Importação de Extrato Bancário. Essas entidades apresentavam posições acessórias no DFD, porém agora estão em posição de destaque no Diagrama de Classes.

- Os processos do DFD, na figura 27 do Anexo B, "Consistir Lançamentos de Extrato Bancário", "Efetivar Importação de Extrato Bancário", "Atualizar Saldo Banco", "Consultar Lançamentos Importados", "Consultar Saldo Banco", "Emitir Relatório de Ocorrências na Importação", são agora as operações do Diagrama de Classes da figura 28 do Anexo B. Tal como a inversão de posicionamento e importância das entidades entre o DFD e o Diagrama de Classes, ocorre também com os processos que estavam em destaque no DFD e agora estão em segundo plano, em relação aos dados dentro das classes.

- Nem todos os depósitos de dados do DFD foram representados em classes, pois eram entidades fracas, tais como: Arquivo TXT do banco, Controle de Importação, Valores Base.

- Atenção com o conceito de classe, pois ele é mais amplo, e nem sempre a correspondência entre depósitos de dados e classes é direta. Por exemplo, no DFD estão representados dois depósitos de dados: Tabela de Códigos de Irregularidade e Irregularidades Ocorridas na Importação, ou seja, o primeiro são as irregularidades possíveis de ocorrer em uma importação, o segundo são as irregularidades ocorridas em uma importação. No Diagrama de Classes, a classe Irregularidades envolve ambos os conceitos. A diferenciação desse exemplo é detalhe de implementação e o Diagrama de Classes trabalha com conceitos.

Os processos são chamados uns pelos outros, mas quem são os responsáveis por eles? O DFD parece um imenso programa que está gerenciando tudo, que atende a diversas solicitações vindas de diversos pontos do sistema. Porém, quando apenas uma entidade é responsável por muitas atribuições, fica mais difícil gerenciar.

Entretanto, no Diagrama de Classes, cada classe é responsável por um conjunto de operações em si encapsuladas.

Embora o DFD e o Diagrama de Classes sejam diagramas distintos pela sua natureza, onde um é dinâmico e o outro é estático, estes foram comparados para mostrar a inversão, pois ambos podem ser utilizados como ponto de partida para a Especificação de Requisitos de Software, levando em consideração cada um a sua técnica.

Para o especialista em Análise Estruturada, ao fazer o Diagrama de Classes, é importante raciocinar de forma semelhante ao fazer o Diagrama de Entidade-Relacionamento, determinando as entidades fortes do negócio, porém lembrando sempre que são mais do que apenas um conjunto de dados que irão defini-las como classes, e sim o comportamento também, que podem ser especificados

nesse momento ou depois quando for modelado o Diagrama de Seqüência. Nesse momento, não devem ser consideradas as entidades de implementação.

As associações e as multiplicidades especificadas em um modelo de classes, também são semelhantes às cardinalidades especificadas em um DER.

Outra importante conclusão é que, enquanto na Análise Estruturada há visões dissociadas entre a decomposição funcional e a modelagem de dados, ou seja, os processos dos dados, na Orientação a Objetos, os dados e as funções são vistos de forma agregada, não ocorrendo uma modelagem separada para cada um desses componentes. Portanto, também sob esse ponto de vista a Orientação a Objetos é mais natural, pois processos e dados não se encontram separados em nenhuma atividade real.

No que se refere à avaliação do grau de dificuldade em elaborar o Diagrama de Classes, podemos concluir que é mais difícil que o DFD e mais fácil que o DER. O DFD é elaborado como se o analista estivesse contando uma história em desenhos, diagramas. Ora, contar histórias e acontecimentos é atividade que as pessoas fazem todos os dias, a partir da ordem em que os fatos ocorrem. E quando se tratam de atividades desempenhas repetidamente com freqüência, assim como contadas por quem desempenha essas atividades, fica mais fácil ainda, precisando apenas representá-las seguindo uma notação pré-estabelecida.

Contudo, o Diagrama de Classes requer um trabalho de observação; as pessoas também observam o mundo e as coisas ao seu redor a todo o momento, porém não é apenas a observação que o Diagrama de Classes requer, é preciso análise das entidades que são importantes para o negócio, como poderão interagir entre si. Então, esta é uma observação com análise crítica.

O Diagrama de Classes é mais fácil que o DER, pois não é necessário abordar detalhes de implementação, ou seja, as entidades fracas, pelo menos nessa fase de especificação de requisitos.

Conclui-se nessa etapa da Análise de Requisitos da Análise Estruturada, que os processos possuem uma modelagem própria e os dados também, ou seja, o DFD e o DER respectivamente. O desenvolvimento do *software* é baseado em duas abordagens distintas: a decomposição funcional e a modelagem de dados. Nota-se, então, visões dissociadas em que se distinguem os processos de dados. Entretanto, em passos a diante na modelagem, essas duas abordagens terão que se cruzar, para espelhar a realidade.

No paradigma orientado a objetos, os dados e as funções são vistos de forma conjugada logo no início da modelagem.

O Diagrama de Classes possibilita o desenvolvimento incremental que é mais difícil na Análise Estruturada. A seguir, poderá ser observado, pelos demais artefatos em OO, que todos dependem do Diagrama de Classes para serem elaborados, e todos, também, o complementam de alguma forma. É interessante que o Diagrama de Classes forneça uma base única para qualquer alternativa de solução futura que possa ser dada ao sistema. Por exemplo, se será utilizado em *desktop*, pela *Internet*, ou celular, não importa se utiliza uma ou todas essas opções, o Diagrama de Classes servirá de base para o desenvolvimento de todos os tipos de solução.

3.1.1.2 Desenvolvimento do Diagrama de Casos de Uso

Os casos de uso são artefatos excelentes para aprimorar a compreensão dos requisitos. Segundo Craig Larman (2000,p.68), os casos de uso não são exatamente uma especificação de requisitos, mas ilustram e implicam requisitos na história que eles contam.

Um caso de uso é uma narração que mostra a seqüência de passos de um ator (um agente externo que pode ser uma pessoa usuária do sistema ou outro sistema, dispositivo) que usa um sistema. Um caso de uso expandido é bem detalhado acerca das ações dos atores e do que o sistema deve executar, sendo

bastante útil para obter uma compreensão bem profunda dos processos e requisitos. Eles são freqüentemente executados em um estilo "conversacional" entre os atores e o sistema (Wirfs-Brock93 apud Larman).

É muito fácil desenvolver um caso de uso, é como narrar um "acontecimento". Efetuar uma narração não é habilidade exclusiva de profissionais de TI. Por isso, os casos de uso são fáceis do usuário entender.

Ao descrever casos de uso, deve-se considerar como o sistema deverá funcionar, ou seja, pensar na solução futura; se a tecnologia concreta de entrada e saída e sua implementação geral, já tiver sido determinada, melhor ainda, pois já é possível utilizá-la na descrição. Se, durante a especificação de requisitos a técnica de prototipação das *interfaces* gráficas for trabalhada, o caso de uso poderá incluir diagramas de esboço das telas envolvidas, bem como discussão da interação de nível baixo com os *"widgets"* da interface. Entretanto, se ainda não tivermos isso determinado, essas especificações poderão ser detalhadas nos diagramas da fase de projeto.

Uma grande vantagem dos casos de uso é a delimitação das "fronteiras" do sistema, ou seja, o que realmente é de responsabilidade do sistema fazer, o que faz parte do escopo do projeto. É importante estabelecer isso o quanto antes para evitar controvérsias com os clientes futuramente, durante a implantação e evitar questionamentos, tais como: "Mas o sistema não contempla isso?"

Pode-se dizer que o Caso de Uso é o instrumento que mais aproxima a técnica Estruturada à da Orientação a Objetos; é como se fosse a ponte para conduzir de uma a outra, pois o Caso de Uso foi introduzido à técnica OO pela grande falta que profissionais sentiram do DFD na descrição detalhada de processos, e o Caso de Uso apresenta semelhanças com DFD's detalhados.

O suporte para a OO que o Caso de Uso fornece é o refinamento do Modelo de Classes em termos da descoberta de mais algumas entidades fortes de

negócio, que ainda não tinham sido descobertas, e o detalhamento dos atributos das classes.

3.1.1.3 Desenvolvimento do Diagrama de Seqüência

O aspecto dinâmico do sistema começa a ser enfatizado nesse momento. Para cada cenário determinaremos as classes envolvidas cujos objetos participarão da interação, que ocorre, basicamente, através de troca de mensagens.

De certa forma, pode-se tomar como base a técnica de elaboração do DFD, no sentido de estar relacionando as funcionalidades dos processos do sistema, só que em um nível de detalhe mais amplo, como no DFD detalhado, ao nível de troca de mensagens; especificando ou não, fluxo de dados.

A similaridade entre o diagrama de seqüência e o DFD pode ser observada na determinação dos processos, porém uma grande diferença entre o DFD e o Diagrama de Seqüência é que no primeiro a inteligência está distribuída pelo sistema e não se sabe quem é responsável por qual ação; já na criação do Diagrama de Seqüência deve-se ter bem claro o princípio de atribuição de responsabilidades. Isto se deve ao fato de que a mensagem que um objeto recebe ser uma requisição para que ele execute um ou mais de seus métodos, ou delegue para outro(s) objeto(s) através do envio de outra(s) mensagem(ns), ou seja, para executar uma determinada ação de um processo é preciso que seja enviada mensagem ao objeto que será responsável por essa ação.

Além dos aspectos acima mencionados, cabe aqui ressaltar um paralelo muito interessante entre o Diagrama de Seqüência e os princípios de Administração, dentre os quais, o trabalho em equipe, essencial nos dias de hoje, no qual cada membro da equipe contribui tendo objetivos em comum para atingir os resultados esperados pela gerência. Desta forma, os objetos do diagrama de seqüência são a equipe de trabalho, cada qual desempenhando seus métodos, que são as tarefas, solicitadas por outros objetos, outros membros da equipe, para atingir os

resultados esperados que o processo deve apresentar, coordenados pelo gerente que é o Analista de Sistemas.

Os diagramas de seqüência são uma boa forma de testar o projeto, uma excelente forma de documentação, e outro aspecto fundamental, são uma forma de detectar gargalos no projeto, pois observando as mensagens que são enviadas para um objeto e por quanto tempo os métodos invocados são executados, pode-se obter uma compreensão de onde seja necessário modificar o projeto para balancear a carga dentro do sistema.

Concluímos, então, que é possível comparar o Diagrama de Seqüências com o DFD, mas destacando a clara atribuição de responsabilidades pelos processos, os quais não ficam "espalhados" como no DFD.

Além disso, o aspecto de desenvolvimento incremental é reforçado, pois o Diagrama de Classes é mais uma vez refinado, acrescentando as operações às classes por elas responsáveis.

3.1.1.4 Desenvolvimento do Diagrama de Estados

Esse diagrama é importante para o aspecto de controle do sistema, que nos auxilia a determinar quais atributos e valores correspondentes serão utilizados para estabelecer o estado das classes.

Esse diagrama deve ser desenhado, em especial, para classes que exibem comportamento complexo que, geralmente, mas nem sempre, dependem de atividade assíncrona; assim como também são úteis em ambientes de tempo real, tipicamente complexos. Se um objeto apresenta diferentes comportamentos, dependendo de seu estado, então, deve-se desenhar o diagrama de estados para ajudar a compreender as diferenças.

Este diagrama apresenta um caráter dinâmico também, que pode ser, parcialmente, baseado no diagrama de seqüência para verificar os processos lá mapeados e os estados pelos quais devem passar os objetos das classes em estudo. Além disso, será utilizado para desenvolvimento do Documento de Testes.

O diagrama de estados é uma importante ferramenta de controle; desenvolvendo o diagrama de estados nessa fase de desenvolvimento, facilitará, posteriormente, o desenvolvimento dos componentes do sistema e seus algoritmos. Pode-se também notar que, o aspecto de desenvolvimento incremental também está presente, pois é importante fazer o mapeamento do diagrama de estados com o diagrama de classes ressaltando os atributos importantes para controle.

3.1.1.5 Conclusão da fase de Requisitos em OO

Nessa etapa da Análise de Requisitos da orientação a objetos, pode-se concluir que:

a) A fase de análise é bem mais detalhada do que na Análise Estruturada, percebida pela diferença da quantidade de artefatos produzida;

b) A modularidade também é desde o início trabalhada na OO, o que na análise estruturada só ocorrerá na fase seguinte de Projeto;

c) Percebe-se a importância do modelo de classes que norteou o desenvolvimento dos demais diagramas da fase de requisitos, e também irá direcionar os diagramas da fase de projeto;

d) As mensagens do diagrama de seqüência partem da interação pré-definida nos relacionamentos do diagrama de classes e os objetos participantes são derivados das classes definidas;

e) Os estados que serão representados em diagramas de estados devem definir as variáveis de controle especificadas nos atributos das classes. Os eventos que provocam as transições de estados, ou são eventos de atores, representados nos casos de uso, ou são mensagens trocadas entre os objetos;

f) Percebe-se o desenvolvimento incremental da técnica e que existe troca de informações entre os artefatos, pois um complementa o outro; por isso, é essencial no desenvolvimento de um projeto usar uma boa ferramenta CASE que mostra esses relacionamentos e a rastreabilidade entre os artefatos;

3.1.2. FASE 2: Projeto

A seguir, serão descritos as atividades desempenhadas e os produtos gerados na etapa de Projeto em OO:

3.1.2.1 Desenvolvimento do MER

É importante ressaltar que o Desenvolvimento do DER **só será necessário para as classes persistentes**, ou seja, as classes cujos objetos são armazenados em um banco de dados relacional.

Na Análise Estruturada, durante a elaboração do DFD em seus refinamentos sucessivos através de decomposição funcional, e a criação de outros fluxos (que são um detalhamento do fluxo macro: diagrama de contexto), começam a surgir pistas sobre os dados requeridos, os quais são elementos de uma estruturação que emprega o Diagrama de Entidade-Relacionamento.

Em Orientação a Objetos, o MER é elaborado a partir do Diagrama de Classes, pois é só um suporte às suas classes persistentes. Nesse ponto, a

preocupação recai sobre os aspectos de implementação física do sistema, e assim, entram em cena as entidades fracas que são úteis apenas por necessidade de implementação, e que por esta razão não foram representadas no Diagrama de Classes.

Eis, então, outra diferença em relação à Análise Estruturada: enquanto na Análise Estruturada o MER é aplicado na fase de Especificação de Requisitos, na Orientação a Objetos o MER é aplicado na fase de Projeto, por ser apenas um suporte para o Diagrama de Classes no mapeamento da estrutura de projeto físico de banco de dados, e mesmo devido ao fato de haver sistemas que nem utilizam Banco de Dados.

No anexo B, figura 29, está o exemplo de DER referente ao mesmo negócio do DFD do exemplo anterior, representado aqui apenas para enfatizar as diferenças entre ele e o diagrama de classes, fator freqüente de confusão entre profissionais da Análise Estruturada.

Como na OO o MER serve apenas de suporte ao Diagrama de Classes, é necessário saber mapear corretamente as **classes persistentes** de um modelo de classes, para um banco de dados relacional. Conforme Page-Jones(2001,p.54), é imprescindível desempacotar as informações dos objetos sob uma forma tabular e normalizada antes de salvá-los.

3.1.2.2 Desenvolvimento do Diagrama de Componentes.

Nessa etapa nascem os pacotes de software em OO, os quais são similares aos módulos gerados na análise estruturada quando é feito o DHF. Os pacotes são organizados por classes, ao passo que os módulos são organizados por funções.

Analisando o diagrama de classes, é necessário visualizar as classes relacionadas com um mesmo propósito e as organizar em pacote. Desta forma, os componentes já nascem com a chance de serem genéricos.

Durante a componentização, os componentes devem ser organizados, considerando não só o negócio, mas também a organização da implementação segundo a arquitetura a ser utilizada. Esse procedimento é o ideal, pois, ao especificar os componentes, deve-se relacionar todos os serviços que serão necessários, por exemplo: JDBC, EJB, no caso de ser utilizada a tecnologia JAVA.

Desta maneira constata-se que o desenvolvimento orientado a objetos é incremental. Para desenvolver componentes, basta acrescentar ao Diagrama de Classes, que até então só continha as entidades de negócio, as entidades de *software* considerando a tecnologia a ser utilizada no ambiente de desenvolvimento: *dll*'s, *storage procedures*, *applets*, etc. Afinal, tudo é um objeto.

Outro fato importante a ressaltar é que a Análise e o Projeto Orientados a Objeto podem ser utilizados mesmo quando o *software* for implementado por uma tecnologia não orientada a objeto, estabelecendo assim, um padrão de desenvolvimento. Isso acontece devido ao fato de que somente na última etapa do desenvolvimento ajusta-se a modelagem à tecnologia empregada. Interessante notar também que, essa abordagem facilita a integração de tecnologias orientadas a objetos com outras que não o sejam, pois nesse momento criam-se componentes para *software* orientado a objetos, como para os que não sejam, e até para os sistemas legados.

Fazendo outra comparação com a Análise Estruturada, conclui-se que nessa, somente na fase de projeto, com a elaboração de DHF, faz-se a modularização do sistema. Conforme dito anteriormente, os dados e funções tinham seus próprios modelos na fase de requisitos da Análise Estruturada e esses modelos tinham que se cruzar em algum momento, sendo esse o momento na análise estruturada, o qual

ocorre bem depois, comparando com a Orientação a Objetos, que possui essa filosofia e a aplica desde o início do desenvolvimento de *software*.

Pode-se afirmar que esses módulos do DHF são precursores dos componentes, sendo esse mais um motivo para tentar diminuir a distância entre as duas metodologias, a Análise Estruturada e a Orientação a Objetos.

3.1.2.3 Desenvolvimento do diagrama de IHM

Os objetos da *Interface* Homem-Máquina nada mais são do que tipos especiais de componentes (telas, formulários), especificados, também, a partir das classes, as quais endereçam telas e documentos de IHM.

Para elaboração das *interfaces* com o usuário, devem ser considerados os processos especificados no diagrama de seqüência e os casos de uso. A partir dessa perspectiva, constata-se que a seqüência de ações no tempo, enfatizada no diagrama de seqüência, ajuda a determinar a ordem na navegação das telas.

Além da reunião de todos os artefatos já desenvolvidos, para não esquecer nenhum detalhe a ser representado na *interface* com o usuário, deve-se elaborar o projeto da *Interface* considerando outros fatores, tais como:

- Perfil dos usuários
- Facilidade de uso das aplicações

É importante também estabelecer um diagrama de navegação das *interfaces*, pois isto leva a um rápido entendimento de como o sistema deve funcionar. Através dessa visão ampla de *interfaces*, é conveniente levantar uma série de questões com o usuário; tais como: "Por que não consigo ir da tela de edição de clientes para a tela de pedidos de clientes, que apresenta uma lista de todos os pedidos que um cliente efetuou?", ou também: "Por que não posso obter o mesmo

tipo de lista do ponto de vista de um produto?". Em alguns casos, pode ser interessante saber quais pedidos incluem um certo produto, especialmente, quando o produto não foi prontamente fornecido ou não está mais disponível.

Essas questões são fundamentais e apresentam-se bem claras após o desenho de um fluxo de navegação de *interfaces*, evitando futuros descontentamentos dos usuários.

3.1.2.4 Conclusões sobre a fase de Projeto em OO

Pode ser assegurado que, pela riqueza de detalhes ocorridos na etapa de análise em OO, a fase de projeto em OO fica mais simples de se efetuar em comparação a de Projeto na análise estruturada.

Na Análise Estruturada, a modularidade é trabalhada a partir desse momento na especificação do DHF, o que a OO já havia começado desde o início do desenvolvimento.

Todavia, sempre existe o *gap* entre a análise e a modelagem, considerando os aspectos de implementação. Com a componentização em OO, esses aspectos ficam confinados na implementação dos componentes, os quais se inspirados na OO podem apresentar um grau de generalidade maior, pois a arquitetura que a OO provê é superior.

A quebra na OO é menor, pois desde o início a modularidade foi criada e vem sendo apenas detalhada e complementada.

Mais uma vez, o Diagrama de Classe aparece como a base para o desenvolvimento incremental na fase de projeto, pelo fato da **classe trazer várias dimensões de implementação: Apresentação, Negócio e Persistência.**

Pode-se observar na especificação das telas, que está ocorrendo a manifestação física da implementação das classes em forma de IHM (formulários, telas, relatórios).

3.1.3 FASE 3: Implementação

Essa fase contempla a codificação propriamente dita, e é interessante detalhar o Diagrama de Classe, complementando-o com a determinação da visibilidade de atributos e métodos, a assinatura e os algoritmos de implementação dos métodos e definição de *interfaces*.

O ideal após fazer uma Análise e Projeto em OO é implementar o código em uma linguagem orientada a objetos, se possível; porém, essa decisão depende da arquitetura utilizada pela empresa e da capacitação dos profissionais na tecnologia a ser adotada. Entretanto, mesmo que a implementação seja feita utilizando recursos físicos não orientados a objetos, a Análise e o Projeto Orientados a Objetos é pertinente, pois o sistema terá mais chances de ser robusto, genérico e flexível.

3.1.3.1 Conclusões sobre a fase de Implementação em OO

De acordo com os dados esboçados até então, pode-se elucidar que a técnica OO apresenta maior transparência na passagem da fase de modelagem para a de construção através da introdução de detalhes, não requerendo uma reorganização do modelo. Trata-se, assim, de um desenvolvimento incremental.

Por fim, o que se pode destacar na Orientação a Objetos é a proeza de ter unificado os formalismos utilizados na análise, projeto e programação. Os conceitos envolvidos são os mesmos, independentemente da fase do desenvolvimento do *software*, o que desta maneira, vem facilitar a forma de comunicação entre o pessoal técnico envolvido no projeto.

3.2 Avaliação da Técnica

O resultado deste estudo está exposto na figuras 15 e 16, a seguir; a figura 15 apresenta os artefatos gerados em cada atividade do processo de desenvolvimento de *software* em OO, explicando as vantagens, porém, expondo certas dificuldades ao elaborar esses produtos.

A figura 16 apresenta o inter-relacionamento entre todos esses artefatos; o quadro da figura 16 deve ser lido da seguinte forma: os diagramas expostos na horizontal dão contribuições para o desenvolvimento dos diagramas expostos na vertical; por exemplo, a intersecção entre diagrama de casos de uso e o diagrama de classes expõe que os casos de uso ajudam a complementar o diagrama de classes com atributos e operações que não haviam sido previstas anteriormente, ou seja, foi colocada uma contribuição do diagrama de casos de uso ao desenvolvimento do diagrama de classes.

Produtos	Vantagens	Dificuldades
Documento de Requisitos	Minimizar as possibilidades de fracasso do projeto de sistemas	Entender exatamente as necessidades do usuário
	Minimizar custo com a antecipação de possíveis erros no desenvolvimento	Definir corretamente o escopo do projeto, delimitando as fronteiras do sistema
Diagrama de Classes	Oferece a base para o desenvolvimento incremental de todas as etapas posteriores, estabelecendo a vantagem de OO em não havendo quebra de estruturas no decorrer do desenvolvimento	Diferenciar as entidades de negócio das entidades que apenas servirão de suporte para implementação
	Ajuda a modularizar o software, pois é um dos primeiros a ser elaborado, apresentando as entidades fortes do negócio como classes e todos os artefatos posteriores são em torno delas designados.	Especificar o número adequado de classes em um diagrama de classes, não criando classes em demasia, e nem classes enormes com muitas responsabilidades.
	Ajuda a dar estabilidade ao sistema, pois assim como a estrutura de um edifício que dá alicerce, o Diagrama de Classes dá alicerce ao sistema inteiro.	
Diagrama de Casos de Uso	Fornece um instrumento, compreensível tanto ao usuário quanto ao profissional de TI , não ambíguo, que ajuda a definir e descrever com precisão as funcionalidades do sistema, com a visão do usuário.	Especificar a quantidade correta de requisitos que serão representados por um caso de uso, não agregando muitos requisitos em um único caso de uso, o que dificultará possíveis manutenções.
	Base para testes e validação de todo o projeto	
	Fácil de ler	
	Fácil de modelar	
	Ajuda a definir as funcionalidades do sistema a partir do ponto de vista do usuário.	
	Ajuda a delimitar as fronteiras do sistema: os Casos de Uso são a parte interna e os atores são a parte externa	
Diagrama de Seqüência	Permite a validação de passos anteriores, como por exemplo, Diagrama de Classes pois pode ser detectado a falta de determinadas operações especificadas atribuídas às Classes. Pois o Diagrama de Interação ajuda a mapear as funcionalidades do sistema pois é um dos Diagramas da Visão Dinâmica da UML.	A parte mais difícil é determinar todos os objetos que interagem em um cenário, e enviar as mensagens para os objetos adequados realizarem a execução das operações.
	Valida as associações existentes no Diagrama de Classes	
	Auxilia na distribuição de responsabilidades por operações em cada classe	
	Facilita a compreensão do fluxo global de controle das aplicações	
DER		Fazer o mapeamento das classes persistentes de um Diagrama de classes para o DER.
Diagrama de Componentes	Auxilia na modularidade e flexibilidade do sistema	Identificar as similaridades e critérios para reunir as classes em componentes
IHM	Facilita a validação do funcionamento do sistema com o usuário, através de prototipação.	Desenvolver uma interface que se adeque a todos os requisitos do usuário: perfil, capacitações, necessidades.

Figura 15 - Vantagens e dificuldades na elaboração dos artefatos de software em OO

Relacionamento entre Diagramas	Documento de Requisitos	Diagrama de Classe	Diagrama de Casos de Uso	Diagrama de Seqüência	Diagrama de Estado	DER	Diagrama de Componentes	IHM
Documento de Requisitos	———	Deve refletir as entidades fortes do negócio que podem ser identificadas no Documento de Requisitos	Os Casos de Uso expressam os requisitos obtidos com os usuários São uma ferramenta de comunicação e validação dos requisitos.	O Diagrama de Seqüência deve refletir as operações que as classes devem executar de forma a atender os requisitos funcionais	O Diagrama de Estado deve refletir os estados pelos quais as classes devem passar de forma a atender os requisitos funcionais	O DER deve refletir a especificação de requisitos não funcionais, como por exemplo: performance.	Os componentes de software devem ser determinados pelo agrupamento de classes similares (req.funcionais) e a arquitetura definida para o cliente(req.não funcionais).	O IHM deve refletir os requisitos de usabilidade desejados e adequados aos perfis dos usuários.
Diagrama de Classe	Fornece a base para esboçar um diagrama de classes.	———	Os casos de uso ajudam a complementar os diagramas de classe com atributos e operações que não haviam sido previstos anteriormente. Validar o Diagrama de Classe no sentido de eliminar classes de implementação, evitando a confusão freqüente entre o DER e o Diagrama de Classes. Isso pode ser feito eliminando classes que não tenham referência em nenhum Caso de Uso.	Os diagramas de Seqüência ajudam a complementar o Diagrama de Classe, atribuindo as operações às classes responsáveis. Contribui para validar as assinaturas de um Diagrama de Classe, pois a troca de mensagens entre objetos indica que deve existir uma associação entre as classes correspondentes a esses objetos.	Esse diagrama deve ser desenhado, em especial, para classes que exibem comportamento complexo que, geralmente, mas nem sempre, dependem de atividade assíncrona; nem sempre também são úteis em ambientes de tempo real, tipicamente complexos. Se um objeto apresenta diferentes comportamentos, dependendo de seu estado, então, deve-se desenhar o diagrama de estados para ajudar a compreender as diferenças	Serve de suporte às classes persistentes do Diagrama de Classe; é a manifestação física das classes implementadas na camada de persistência.	Expressa a manifestação física das classes em forma de componentes físicos implementados na camada de negócio do software.	Expressa a manifestação física das classes em forma de interface com usuário, implementadas na camada de apresentação.
Diagrama de Casos de Uso	Os requisitos obtidos com os usuários são a base para a elaboração dos diversos cenários de caso de uso.	O Diagrama de Classe pode ajudar a identificar classes importantes de negócio que não tenham sido referenciadas por nenhum Caso de Uso, evitando possíveis esquecimentos.	———	Expõe os objetos envolvidos para execução do Use Case, ou seja, o Diagrama de Seqüência pode ser usado para demonstrar a interação dos objetos em um Caso de Uso. Um Diagrama de Seqüência adiciona a dimensão de tempo para as interações dos objetos, expressando o Use Case de forma mais precisa e técnica.	Utilizar o Diagrama de Estados para visualizar o comportamento de um objeto para diversos Casos de Uso. Consegue-se, desta forma, entender o relacionamento entre Casos de Uso	As entidades, atributos e relacionamentos do DER devem refletir fidedignamente as informações registradas nos Casos de Uso.	Os serviços dos componentes devem operacionalizar as ações especificadas nos Casos de Uso.	A elaboração de protótipos facilita a descrição dos Casos de Uso.
Diagrama de Seqüência	Os requisitos funcionais são a base para a elaboração dos diversos cenários expressos pelos Diagramas de Seqüência.	As classes são modelos para a criação de objetos(instâncias) do Diagrama de Seqüência.	Serve de roteiro para elaboração do diagrama de seqüência.	———	Identificar se foram relacionadas as mensagens para a execução dos métodos que vão provocar a transição de estados, evitando possíveis esquecimentos.	Os relacionamentos do DER devem ser compatíveis com as interações registradas nos Diagramas de Seqüência.	Os serviços dos componentes devem operacionalizar as ações especificadas nos Diagramas de Seqüência.	A elaboração de protótipos facilita a especificação das trocas de mensagens ocorridas nos Diagramas de Seqüência.
Diagrama de Estado	Os requisitos funcionais também devem ser considerados para a elaboração do Diagrama de Estados, determinando a seqüência de estados pelos quais as classes devem passar, para atender aos requisitos acordados com o cliente.	Ajuda a determinar quais são as classes que deverão possuir um diagrama de estado, por exibirem comportamento complexo, percebido pelo Diagrama de Classes, pela grande quantidade de atributos e associações com outras classes. Os atributos do Diagrama de Classe definem o estado do objeto em determinado momento, enquanto que os métodos definem as transições que esses estados tomam.	O conjunto de Casos de Uso quer influenciam as classes que tenham seus estados mapeados, devem servir de base para a elaboração do Diagrama de Estado, para determinação da seqüência dos eventos que devem ocorrer para a transição desses estados.	Ajuda a determinar quais são as classes que deverão possuir um diagrama de estado, por exibirem comportamento complexo, visualizado pelo Diagrama de Seqüência pela grande quantidade de mensagens recebidas e/ou enviadas	———	Atributos do controle de entidades do DER devem ser responsáveis pela retenção de estados registrados nos Diagramas de Estados.	Os serviços dos componentes devem operacionalizar as ações que provocam a transição de estados nos Diagramas de Estado.	A elaboração de protótipos facilita a determinação dos pontos responsáveis pela transição de estados nos Diagramas de Estado.
DER	Os requisitos funcionais são fundamentais para a determinação das entidades e seus relacionamentos no DER; além disso, os requisitos não funcionais devem ser considerados, tais como desempenho, escalabilidade, segurança.	Serve de ponto de partida para a especificação do DER, devendo ser feito o mapeamento das classes persistentes para o DER.	Ajuda a determinar as entidades, seus atributos e relacionamentos, através das informações descritas nos vários cenários dos Caso de Uso	Auxilia em determinar relacionamentos entre as entidades, que podem ser visualizados pela troca de mensagens entre os objetos. Vale ressaltar que essa regra vale para objetos de classes persistentes.	É responsável por determinar quais atributos serão responsáveis pela retenção de estados de objetos de classes persistentes.	———	Os componentes de software devem possuir atributos encapsulados que devem corresponder a atributos do DER, sempre quando for necessário o serviço de persistir objetos	A elaboração de protótipos auxilia na determinação de atributos, que refletem os campos da Interface Homem-máquina
Diagrama de Componentes	Os componentes devem ser determinados pelo agrupamento de classes afins, que tenham responsabilidades comuns em relação aos requisitos	Ajuda a determinar os componentes, através da fácil visualização das classes afins que podem ser verificadas pelo Diagrama de Classes.	Através dos Casos de Uso, fica mais fácil visualizar as funcionalidades afins, e sendo, também, relativamente fácil identificar as classes envolvidas em um caso de uso, fica mais simples associar as classes em comum para gerar os componentes.	O Diagrama de Seqüência também auxilia no sentido de expor as funcionalidades que deverão ser suportadas pelos Componentes de Software	O Diagrama de Estados auxilia em demonstrar processos de controle que os componentes devem suportar.	Atributos especificados no DER devem refletir atributos encapsulados nos componentes de software.	———	As funcionalidades visíveis pela IHM, facilitam a especificação dos serviços dos componentes, de forma à atendê-las.
IHM	Os requisitos de usabilidade devem nortear a elaboração da IHM, observando as características adequadas ao perfil de cada grupo de usuários.	Os objetos da *Interface* Homem-Máquina nada mais são do que tipos especiais de componentes (telas, formulários), especificados, também, a partir das classes, as quais endereçam telas e documentos da IHM	Serve de roteiro para elaboração do IHM.	A seqüência de ações no tempo, enfatizada no diagrama de seqüência, ajuda a determinar a ordem na navegação das telas	O Diagrama de Estado ajuda a desenhar uma estrutura de controle para a interface, para que seja seguida a sequencia de passos correta para desenvolvimento das tarefas.	O DER deve refletir as informações de objetos persistentes em suas manifestações físicas na IHM.	Os componentes de software devem prestar os serviços que tornam possíveis as interações e funcionalidades expressas na IHM.	———

Figura 16 - Relacionamento entre Diagramas UML

3.3 FAQ sobre OO

Para desmistificar o paradigma da Orientação a Objetos, estão relacionadas a seguir, algumas questões a serem elucidadas a fim de esclarecer alguns possíveis equívocos da Tecnologia de Objetos:

1. **Dada as vantagens da Tecnologia de Objetos em relação a outras abordagens de Desenvolvimento de Software, por que muitas empresas ainda não adotaram a Tecnologia OO ?**

   Um dos motivos, como coloca Arídio Silva(2002), é que apesar dos princípios da Orientação a Objetos serem simples de entender, "eles são freqüentemente mascarados e dificultados pelo jargão técnico, que parece impenetrável para muitas pessoas.". Esse obstáculo deve ser removido através da desmistificação de conceitos que são elementares para entendimento da técnica.

   Outro impedimento ao uso efetivo da tecnologia de objetos, é que muitos desenvolvedores são geralmente treinados nas ferramentas de OO, sem primeiro terem sido envolvidos numa estrutura apropriada para entendimento dos aspectos conceituais e idéias centrais que formam a tecnologia de objetos. Isso é deplorável considerando que os grandes benefícios dos objetos não podem ser obtidos e disseminados, se os princípios chaves não forem compreendidos na medida necessária. Sem o entendimento do mundo dos objetos é impossível trabalhar com as novas tecnologias da área de informática e da Internet.

2. Quais os critérios para se definir as classes de objetos a partir do entendimento do domínio do problema ?

Tente raciocinar da seguinte forma:
Desde criança, no colégio você certamente estudou funcionalidades, comportamento e estrutura de vários sistemas do corpo humano e deve ter mapeado um sistema completo: o sistema digestivo por exemplo. Só para você relembrar e ver como era simples e elementar, o descrevemos abaixo:

Antes de apresentar a dinâmica do processo de digestão, eram apresentados primeiro os órgãos do sistema e sua composição: A boca, o esôfago, o estômago, o intestino, etc.

Tendo o conhecimento de quem é quem nesse sistema , você só precisava compreender como era o seu funcionamento. A boca mastiga o alimento, e o passa para o esôfago, e assim por diante. Em cada passagem pelos órgãos, alguma ação era efetuada pelo alimento.

Onde estou querendo chegar, é que podemos pensar em um sistema de *software* como o sistema digestivo. Basta identificar as classes, que são como os órgãos, e as responsabilidades de cada uma, que são como as funções de cada órgão.

Se fosse pedido para que você identificasse no sistema digestivo quem são os órgãos e suas ações, você facilmente os identificaria; então, por que não fazer a mesma identificação com os sistemas de *software*, por que complicar? Imagine que um processo a ser desempenhado, por exemplo, emitir um pedido, seja como o processo a ser desempenhado pelo sistema digestivo, que é efetuar a digestão de um alimento, e identifique quem vai colaborar com suas

ações para que o trabalho seja feito.

As mensagens enviadas para que cada objeto desempenhe sua atividade é similar à passagem de alimento de um órgão para outro dentro do sistema digestivo.

Pense naturalmente.

Na análise estruturada há quebra no pensamento, primeiro pensamos em funcionalidades para depois verificar os dados que atendem a funcionalidade.

Ocorre uma interrupção e fica mais difícil de visualizar quem é responsável por qual atividade. Na OO há uma forma natural e direta de pensar, e não artificial.

3. **Qual a diferença entre evento e mensagem ?**

Evento é qualquer acontecimento que ocorre no sistema, um aviso do sistema operacional que um botão foi pressionado, uma mensagem que um objeto enviou para outro.
Desta forma, uma mensagem é um tipo de evento.

4. **O Polimorfismo existe sem a Herança? Qual a relação entre um e outro ?**

Existem vários tipos de polimorfismo, podem ser combinados ou não com o mecanismo de herança, ou seja, a herança pode fornecer o aparato necessário para tornar certos tipos de polimorfismo possíveis. A hierarquia de uma herança forma relacionamentos com capacidade de substituição entre os subtipos e seus progenitores. A hierarquia é provida pela herança e a substituição, pelo polimorfismo. Mas essa situação é apenas um exemplo de polimorfismo: o de Inclusão.

Conforme visto no capítulo 1, existem 4 tipos de polimorfismo.

A seguir, estão relacionados exemplos de Polimorfismo extraídos da obra "Aprenda Programação Orientada a Objetos em 21 dias" do autor Anthony Sintes(2002,p.126) :

Os trechos indicados a seguir, são exemplos de código em Java.

1) Polimorfismo de inclusão

Ao invés de escrever os métodos abaixo:

```
Public void makeSpeak ( PessimisticObject obj ) {
        System.out.println (obj.speak() );
Public void makeSpeak ( OptimisticObject obj ) {
        System.out.println (obj.speak() );
Public void makeSpeak ( IntrovertedObject obj ) {
        System.out.println (obj.speak() );
```

A capacidade de substituição e o polimorfismo de inclusão permitem que se escreva um único método para manipular todos os tipos de objetos PersonalityObject:

```
Public void makeSpeak ( PersonalityObject obj ) {
        System.out.println ( obj.speak() );
}
```

A capacidade de substituição permite que se passe qualquer objeto PersonalityObject para o método e o polimorfismo garante que o método correto seja chamado na instância. O polimorfismo chamará o método com base no tipo verdadeiro da instância (PessimisticObject, OptimisticObject ou

IntrovertedObject) e não em seu tipo aparente (PersonalityObject).

O polimorfismo de inclusão torna mais fácil adicionar novos subtipos em uma aplicação, pois não precisará adicionar um método específico para o novo tipo. Simplesmente, pode reutilizar makeSpeak().

2) Polimorfismo paramétrico

Suponha uma situação em que se queira um método que some dois tipos de argumentos do mesmo tipo. O polimorfismo paramétrico permite que escreva um método para somar todos os tipos.

```
Add( [T]  a, [T]  b ) : [T]
Return a + b;
```

Declarando um método dessa maneira, adia a definição do tipo dos argumentos até o momento da execução.

3) Sobreposição

```
Public class MoodyObject {
  // retorna o humor
  protected String getMood() {
        return "moody";
  }

  // pergunta ao objeto como ele se sente
```

```
        public void queryMood() {
            System.out.println("I feel " + getMood() );
        }
    }

    public class HappyObject extends MoodyObject {

    // redefine o humor da classe
    protected String getMood() {
            return "happy";
    }

    // especialização
    public void laugh() {
            System.out.println(" AHAHAHAH ! ");
    }
}
```

Quando o método queryMood() em HappyObject, o polimorfismo garante a chamada da versão sobrescrita de getMood() em HappyObject, internamente.

4) Sobrecarga

```
Public static int max(int a,int b);
Public static long max(long a,long b);
```

Os métodos max() são todos exemplos de sobrecarga, diferem apenas no tipo de parâmetros. Dependendo dos parâmetros da mensagem que o objeto que está chamando o método max , uma variante ou outra do método será executada.

5. Em um projeto de sistema, utilizando OO, como e em que fase do projeto o Polimorfismo é representado?

Na implementação dos componentes, quando forem detalhados os métodos com as respectivas assinaturas e analisados os algoritmos.

6. O que difere um diagrama de classes de um diagrama Entidade-Relacionamento, dada a grande similaridade entre os dois? Qual a diferença entre uma classe e uma entidade?

O foco do Diagrama de Classes são os conceitos e as aplicações e funcionalidades do sistema, ao passo que o foco do Diagrama de Entidade-Relacionamento são os dados. O Diagrama de Classes é mais amplo que o DER, pois além das classes persistentes que também podem ser encontradas especificadas no DER, há classes de *interface* e classes de controle de processos.

A similaridade entre os dois diagramas é em relação às classes persistentes que se referem às entidades do DER, onde cada atributo das classes tem a correspondência com os atributos das entidades do DER e as associações/ multiplicidades entre as classes têm correspondência entre as cardinalidades entre as entidades do DER.

Mesmo com relação às classes persistentes, nem sempre vale a regra de uma classe ser mapeada para uma tabela, muitas vezes uma classe pode ser representada por várias estruturas de banco de dados relacional.

O DER apresenta aspectos de implementação que não são contemplados em um diagrama de classes realizado na etapa de análise de requisitos.

Já o diagrama de classes que auxilia a implementação das classes de

software apresenta classes de *interface* e controle de processos que pode não ter relação alguma com qualquer tabela ou estrutura que tenha sido definida em um DER.

Há sistemas que nem base de dados utilizam, então não deverá ser realizado o DER, já o modelo de classes deve ser definido sempre para qualquer sistema que seja modelado pela OO.

7. **Qual a importância e a contribuição de cada diagrama UML para a implantação do sistema? Qual o grau de complexidade que deve ser atingido em cada diagrama em cada etapa de desenvolvimento do projeto? Em que ordem elaboramos cada um deles?**

Primeiramente, os diagramas UML facilitam a comunicação entre os profissionais de sistema.

Os diagramas da fase de requisitos não devem contemplar aspectos de implementação.

Por exemplo, o Diagrama de Classes feito na fase de Especificação de requisitos deve contemplar apenas as classes de negócio.

Na fase de implementação, é bom fazer um outro diagrama de classes a partir do diagrama que foi feito na especificação de requisitos, detalhando as assinaturas de métodos, a visibilidade de atributos e métodos, as *interfaces*. É o apoio para a codificação, portanto, enriquecemos com a maior quantidade de detalhes possível.

A ordem de utilização que foi fornecida nesse trabalho foi a seguinte:

1) Modelo preliminar de classes

2) Casos de uso
3) Complemento das classes
4) Diagrama de seqüência
5) Complemento das classes
6) Diagrama de estados
7) DER
8) Diagrama de componentes
9) IHM

Essa ordem é uma sugestão, pois as empresas podem adotar variantes desse esquema de trabalho.

8. Definir e dar exemplos através do Estudo de Caso dos conceitos fundamentais da Orientação a Objetos: Identidade dos Objetos, Polimorfismo, Retenção de Estado, Herança.

Identidade dos objetos

No anexo H, está representado o DER de um módulo que foi modelado no estudo de caso. Algumas classes de negócio tiverem correspondência em tabelas de bancos de dados, por serem persistentes.

O ID de cada objeto será representado pelas chaves primárias das tabelas que foram mapeadas e dão suporte a essas classes.

Exemplo: chave primária OPER_FINANC_ID da tabela OPER_FINANCEIRA representando a classe persistente OperaçãoFinanceira.

Herança

No anexo E, está representado o diagrama de classes do mesmo módulo que foi modelado no estudo de caso. Há a representação de herança entre a superclasse ComponenteCálculo e as subclasses:

ValorBase, ValorDerivado e FunçãoSistema.

Retenção de Estado

No anexo G, está representado o diagrama de estados de uma das principais classes do mesmo módulo que foi modelado no estudo de caso: OperaçãoFinanceira. Há vários estados pelos quais a OperaçãoFinanceira pode se encontrar e esse estado é persistido na base de dados através do atributo COD_SITUAC da tabela OPER_FINANCEIRA.

9. Em que circunstâncias devemos utilizar o Diagrama de Objetos?

Para representar cenários complexos em que o comportamento de uma classe varie de acordo com os variados tipos de objetos que ela possa instanciar.

10. Quais as diferenças entre a definição de Banco de Dados Relacional, Objeto-Relacional e Orientado a Objetos, e qual melhor se enquadraria na Metodologia de Orientação a Objetos?

O artigo "Uma nova era na Tecnologia dos Bancos de Dados" de Carício Afonso Junior(2002), explica muito bem essas diferenças.

A seguir, está exposta a síntese dessas diferenças:

O modelo relacional é baseado no conceito matemático de relação.

Os matemáticos definem como uma relação um subconjunto de um produto cartesiano de uma lista de domínios. Uma tabela é uma

relação, e suas linhas representam relacionamento entre um conjunto de valores.

O modelo relacional é simples, mas fundamentalmente diferente do modelo OO. Bancos de dados relacionais não foram concebidos para armazenar objetos, e programar uma *interface* para armazenar objetos em tabelas não é uma tarefa trivial.

Bancos de dados relacionais não possuem mecanismos para representar características básicas de objetos, como encapsulamento, herança e polimorfismo. Objetos são interligados por referências, tabelas são relacionadas através de chaves primárias e chaves estrangeiras. Um modelo relacional busca normalizar as informações, ou seja, eliminar ao máximo a redundância dos dados armazenados em tabelas, enquanto um modelo OO busca criar objetos que representem o mundo real.

Em geral, duas abordagens são empregadas para armazenar objetos em bancos relacionais: partir da modelagem OO e criar as tabelas para representar os objetos, ou partir de um modelo relacional e criar objetos para representar os processos. A primeira exige programação adicional para refletir as estruturas dos objetos em tabelas, uma situação em que a manutenção do código é complexa e cara. A segunda abordagem pode comprometer totalmente a modelagem da aplicação.

Os bancos objeto-relacionais oferecem suporte aos dois modelos.

Os bancos OO são estáveis, oferecem *performance* e escalabilidade. Armazenar e recuperar objetos de um banco OO é mais rápido do que pulverizar seus atributos em tabelas de duas dimensões.

11. Qual é o futuro da OO e dos Bancos OO ?

Conforme Carício Afonso Junior(2002) cita em seu artigo "Uma Nova Era na Tecnologia dos Bancos de Dados", só no Brasil vários órgãos do governo e grandes empresas dos setores financeiro, elétrico, de telecomunicações e de saúde definiram que vão empregar OO, principalmente JAVA, em todos os novos projetos.

Atualmente, os bancos de dados são predominantemente relacionais, mas uma aplicação OO precisa armazenar objetos.

Os bancos de dados OO evoluíram muito nos últimos cinco anos e atingiram maturidade e *performance* surpreendentes. As novas estratégias de armazenamento e recuperação permitem que esses bancos consigam um tempo de resposta geralmente melhor do que as subseqüentes chamadas a um banco relacional, necessárias para desmontar e remontar objetos.

Mas o modelo relacional foi amplamente empregado nos últimos quinze anos e não podemos simplesmente descartar todas as aplicações já construídas. Esses sistemas devem coexistir pacificamente com as novas aplicações OO. Nesse ponto destacam-se os bancos pós-relacionais. Oferecendo um suporte aos dois modelos, objeto e relacional, esses produtos permitem que todo o investimento nas antigas aplicações seja preservado, enquanto as novas aplicações são construídas utilizando a tecnologia de objetos.

A idéia é evolução, e não revolução, ressalta Afonso Junior.

As aplicações cliente/servidor, desenvolvidas em bancos relacionais, não se adaptam à nova realidade imposta pela *Internet*, naturalmente heterogênea e distribuída.

Armazenar objetos em um banco relacional é uma tarefa difícil. O modelo relacional não possui os mecanismos necessários para representar características básicas do modelo OO. Nesse cenário, para representar as novas aplicações, o banco de dados relacional não traz nenhuma vantagem. A escolha de um banco de dados OO é a evolução natural.

12. O que são esteriótipos?

É uma extensão do vocabulário da UML, permitindo a criação de blocos de construção similares aos já existentes, mas que sirvam para a especificação de um problema específico. Por exemplo, suponha que se queira representar um Hub específico de uma arquitetura que esteja sendo especificada. Ele será representado como um nó estereotipado.

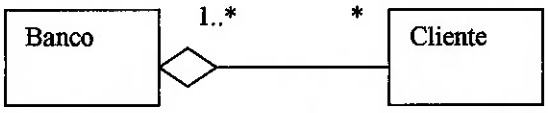
13. Qual a diferença entre agregação e composição?

A composição é um tipo especial de agregação. Na agregação simples, a parte pode ser compartilhada por vários todos. Por exemplo, no modelo de uma casa, uma parede pode ser parte de um ou mais objetos Cômodo. Na composição, por exemplo, em um sistema de janelas, uma Moldura pertence exatamente a uma única Janela.

Além disso, em uma agregação composta, o todo é responsável pela disposição de suas partes, significando que o objeto composto deve gerenciar a criação e a destruição de suas partes.

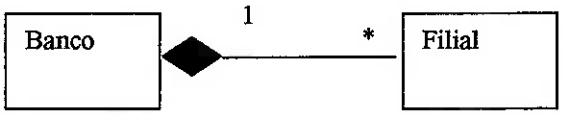
A agregação é um tipo especial de associação e modela um relacionamento "tem um" entre pares. Esse relacionamento significa

que um objeto contém outro. Pares significa que um objeto não é mais importante do que o outro. Importância, no contexto da agregação, significa que os objetos podem existir independentemente um dos outros. Por exemplo:



Nesta figura, um Banco pode conter qualquer número de objetos Cliente. Banco é o objeto que "tem um" no relacionamento.

Já a composição não é um relacionamento entre pares. Os objetos não são independentes uns dos outros. A figura abaixo indica uma composição entre um banco e suas filiais:



Os objetos filial não podem existir independentemente do objeto Banco.

14. O diagrama de classes deve continuar sendo detalhado a ponto de representarmos interfaces, classes ativas, visibilidade de atributos e métodos, polimorfismo? Que detalhe um analista deve chegar para passar uma definição de aplicação para um desenvolvedor?

SIM, todos esses detalhes devem ser especificados na fase de implementação.

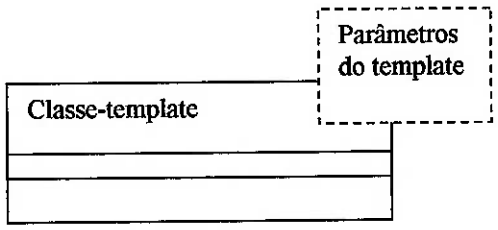
15. Para que servem os templates? Quais as vantagens?

Um template é um elemento parametrizado. Em C++, por exemplo, é possível escrever classes-template, cada uma definindo uma família de classes, e até funções-template. Um template inclui slots para classes, objetos e valores e esses slots servem como parâmetros do template.

O uso mais comum das classes-template é a especificação de recipientes que podem ser instanciados para elementos específicos, tornando-os um tipo-seguro.

A vantagem é possibilitar aplicações genéricas e flexíveis que minimizam a manutenção dos sistemas.

Representação de templates em UML:



4 ESTUDO DE CASO

Como objetivo da parte prática do trabalho, serão desenvolvidas funcionalidades básicas de um sistema de informações da área financeira que contemplam o módulo de captações, a partir das técnicas e métodos explanados no capítulo anterior, os quais compõem a proposta de migração da metodologia de Análise Estruturada para a Orientação a Objetos.

Esta pesquisa foi realizada em uma empresa particular da área de informática, a qual está começando a investir na metodologia de Orientação a Objetos, e por esse motivo, deu apoio e incentivo ao autor dessa monografia para que esse trabalho fosse desempenhado, tanto na disponibilidade das informações, como no oferecimento de oportunidades.

Devido à utilização da Orientação a Objeto, demais funcionalidades poderão ser futuramente acopladas a esse "*Core*" com facilidade, similar a uma linha de produção flexível de carros, por exemplo, na qual encaixamos as partes para montar o produto final.

Será entregue então, o equivalente ao "chassi" do carro no exemplo dado, que se refere à Simulação de Operações Financeiras de Captação do Sistema, que poderá ser utilizada posteriormente em futuras extensões de funcionalidades do sistema como: geração de contratos, simulação de outros tipos de operações tais como aplicações e mútuo, os quais não serão tratados nessa monografia.

Essas funcionalidades básicas deverão ser desenvolvidas utilizando a tecnologia JAVA, escolhida por ser totalmente orientada a objetos, robusta, flexível e multi-plataforma. Portanto, essa escolha influenciará na estruturação dos componentes, na determinação dos serviços oferecidos pela arquitetura.

A seguir, na figura 17, é apresentado um fluxograma que ilustra as etapas de trabalho para elaboração do estudo de caso. Esta é, contudo, uma sugestão de método de desenvolvimento de *software* a ser seguido:

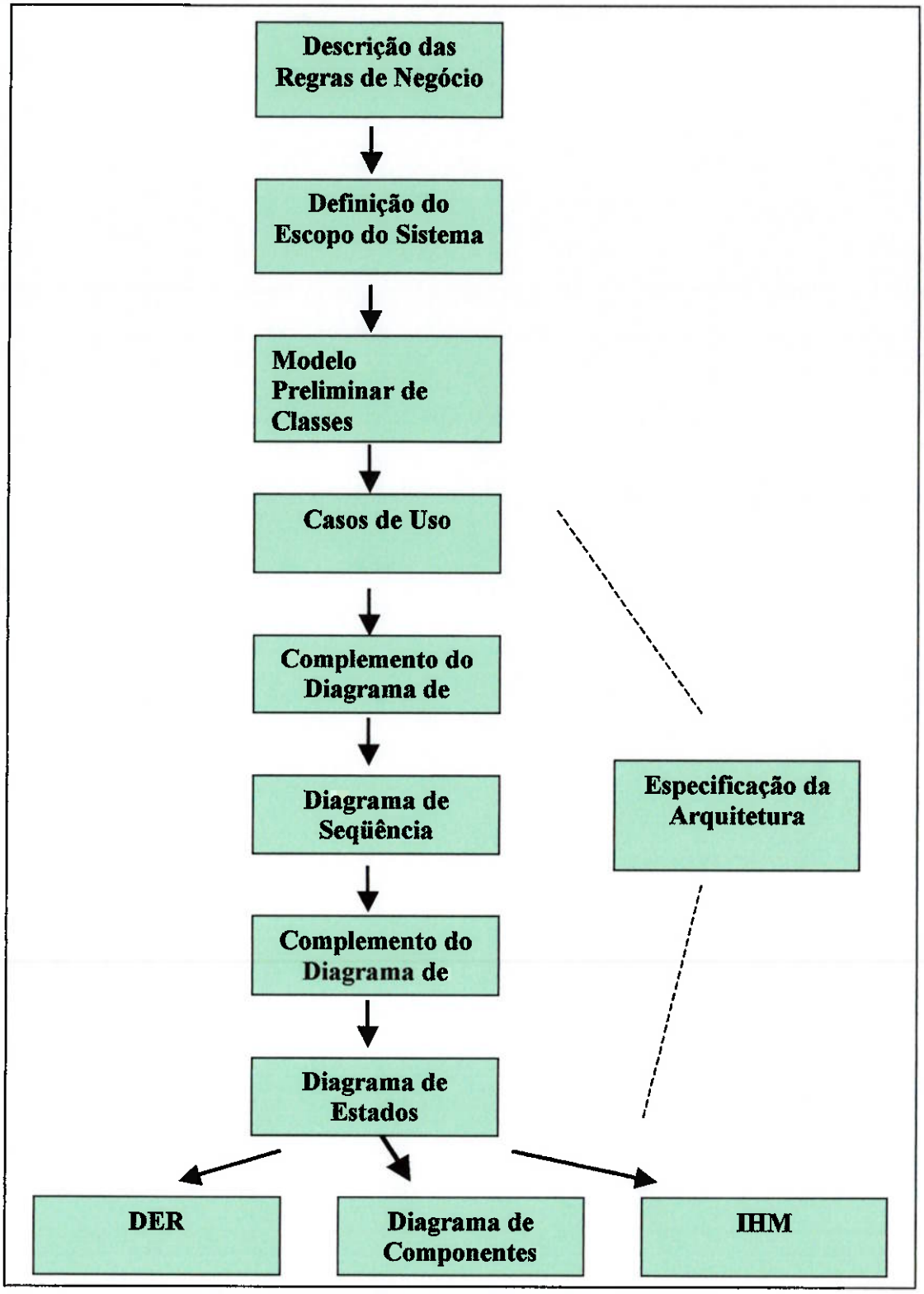


Figura 17 - Etapas do desenvolvimento do estudo de caso

A figura 17 mostra a condução do desenvolvimento desse módulo de sistema envolvendo as etapas especificadas nos retângulos do desenho.

A especificação da arquitetura é uma atividade que pode ser feita em paralelo com a especificação de requisitos, indicada pela linha tracejada do fluxo. Da mesma forma, as atividades de elaborar o DER, diagrama de componentes e o IHM também podem ser elaboradas em paralelo.

4.1 FASE 1: Especificação de Requisitos de Software

4.1.1 Descrição das Regras de Negócio

Como dito anteriormente, os conceitos abordados nesse trabalho serão exemplificados através de um Estudo de Caso, envolvendo o projeto de uma pequena parte de um sistema real. Ao final do projeto, será criado um componente auto-suficiente que comporá um grande sistema.

Para desenvolver esse módulo, em primeiro lugar, serão apresentadas as regras de negócio que esse projeto de *software* focará. Mesmo utilizando metodologia Análise Estruturada é necessário partir desse ponto; então, nessa atividade introdutória, não há diferenças entre as duas metodologias.

Embora seja construído um componente de um grande sistema, é bom ter uma visão geral do negócio em que ele se insere, pois, embora nosso componente deva ser independente e ter vida própria, faz-se relevante ter idéia de futuros relacionamentos que o componente possa ter com outros componentes a serem projetados.

4.1.1.1 Cenário do negócio

A função financeira apresenta-se, atualmente, como importante foco de atenção das organizações, quer atuem na área industrial, de distribuição, serviços, ou mesmo propaganda e marketing, entre inúmeros outros negócios. A obtenção, aplicação e alocação de recursos, de forma eficiente, exige dos executivos, não somente "*expertise*" financeiro e contábil, mas um completo entendimento das diversas áreas da organização. Soma-se a isto, a dimensão temporal da administração financeira que deve estar voltada ao passado, cuja história pode balizar a tomada de decisões, ao presente, quando as ações são executadas, e ao futuro, ao antever dificuldades e oportunidades para iniciar, a tempo, as correções necessárias.

A rápida evolução da tecnologia deve oferecer ferramentas sofisticadas com a finalidade de agilizar a tomada de decisão, sem perda de confiabilidade, o que permite o aproveitamento das melhores oportunidades e o conseqüente aumento de ganhos ou redução de perdas. Assim, os sistemas de gestão financeira devem seguir o rastro desta tendência, não somente como mais uma ferramenta de suporte, mas como agente inovador da gestão financeira voltado à realidade brasileira.

4.1.1.2 A gestão financeira

A gestão financeira tem basicamente três focos de atuação: Tesouraria, Controle de Operações Financeiras e Planejamento Financeiro.

4.1.1.2.1 Tesouraria

A seguir, estão relacionadas funções típicas da área de Tesouraria:

a) Pagamentos:

- Agrupamento de compromissos para pagamento por favorecido e/ou meio de pagamento;
- Pagamento eletrônico padrão FEBRABAN, INTERCHANGE e CITIBANK;
- Desenho e emissão dos meios físicos de pagamento;
- Controle numérico seqüencial dos meios de pagamento, por conta bancária: folhas em estoque, emitidas e canceladas.

b) Movimento Bancário

- Registro dos movimentos de caixa, através das contas bancárias;
- Atualização em tempo real dos saldos bancários;
- Controle dos saldos empresa, banco e conciliado, EDI com banco;
- Visualização em tela dos extratos das contas bancárias.

c) Movimento financeiro:

- Plano de contas financeiro configurado em vários níveis;
- Fluxo de caixa configurável: nível, moeda, empresa, filial, unidade de negócio;

- Pagamentos e recebimentos por natureza financeira;
- De/para conta financeira/conta contábil.

4.1.1.2.2 Operações Financeiras:

A seguir, estão relacionadas funções típicas da área de Operações Financeiras:

- Configuração de moedas, índices econômicos, encargos e modalidades de aplicação e captação;
- Visualização em tela da carteira de aplicação/captação por modalidade;
- Definição de contas contábeis e contas financeiras por operação financeira;
- Registro e controle de cada operação financeira: saldo atual, resgates/liquidações, taxas efetivas e valor de vencimento;
- Geração automática de vencimentos no fluxo de caixa futuro;
- Operações com bancos, empresas coligadas (mútuo) e terceiros;
- Avaliação das operações sobre qualquer moeda: operação, modalidade e portifólio;
- Múltiplas repactuações;
- Operações em qualquer moeda;
- Impressão dos meios de pagamento para aplicações e liquidações de empréstimo.

4.1.1.2.3 Planejamento Financeiro:

A seguir, estão relacionadas funções típicas da área de Planejamento Financeiro:

a) Geração de previsão:

- Importação da previsão de caixa de outros sistemas;
- *Data entry* de previsões pelas áreas de origem;
- Previsão de caixa por empresa em qualquer moeda ou nível do plano de contas financeiro;
- Previsão de caixa dia a dia, semana a semana ou mês a mês.

b) Avaliação da previsão:

- Múltiplas versões de previsão para um período;
- Real x Previsto por grupo, divisão, empresa, filial e unidade de negócio;
- Real x Previsto em qualquer moeda ou nível do plano de contas financeiro;
- Real x Previsto dia a dia, semana a semana ou mês a mês.

c) Suporte à decisão:

- Simulação de cenários de caixa;
- Avaliação de bancos;
- Controle do risco bancário.

4.1.1.3 Tendência

O desempenho eficiente destas funções vem exigindo dos grandes grupos empresariais uma solução centralizada, na qual uma única área atua como Centro Financeiro frente a todas as empresas. Ao desempenhar este papel, a área financeira, além de funcionar como fonte de recursos, passa a gerenciá-los e distribuí-los entre as empresas. As operações das empresas com agentes financeiros externos, tais como bancos, corretoras e distribuidoras são intermediadas por esta estrutura única.

As justificativas para tal abordagem são evidentes:

1) A necessidade financeira de algumas empresas pode ser coberta por empresas do grupo que apresentem disponibilidade de recursos. Operações deste tipo podem significar, simultaneamente, empréstimos a custo reduzido para as primeiras empresas e excelentes aplicações para as últimas.

2) O poder de negociação junto aos agentes financeiros externos aumenta à medida que todo o grupo é visto como um único cliente. Empréstimos a custos menores, aplicações mais rentáveis e outras facilidades decorrentes de tal fato, refletem diretamente no resultado das empresas.

Com base nesta abordagem, a gestão financeira deixa de se referir necessariamente a uma empresa, para incluir um conjunto destas, atuando nas mais diversas áreas.

4.1.2 Definição do Escopo do Sistema

O módulo do Estudo de Caso contemplará apenas o Gerenciamento de Captações, esse será o escopo desse projeto. Por essa razão, será descrito, a seguir, o fluxo de trabalho da área de Captações para que se possa definir os processos de negócio envolvidos a serem gerenciados pelo módulo.

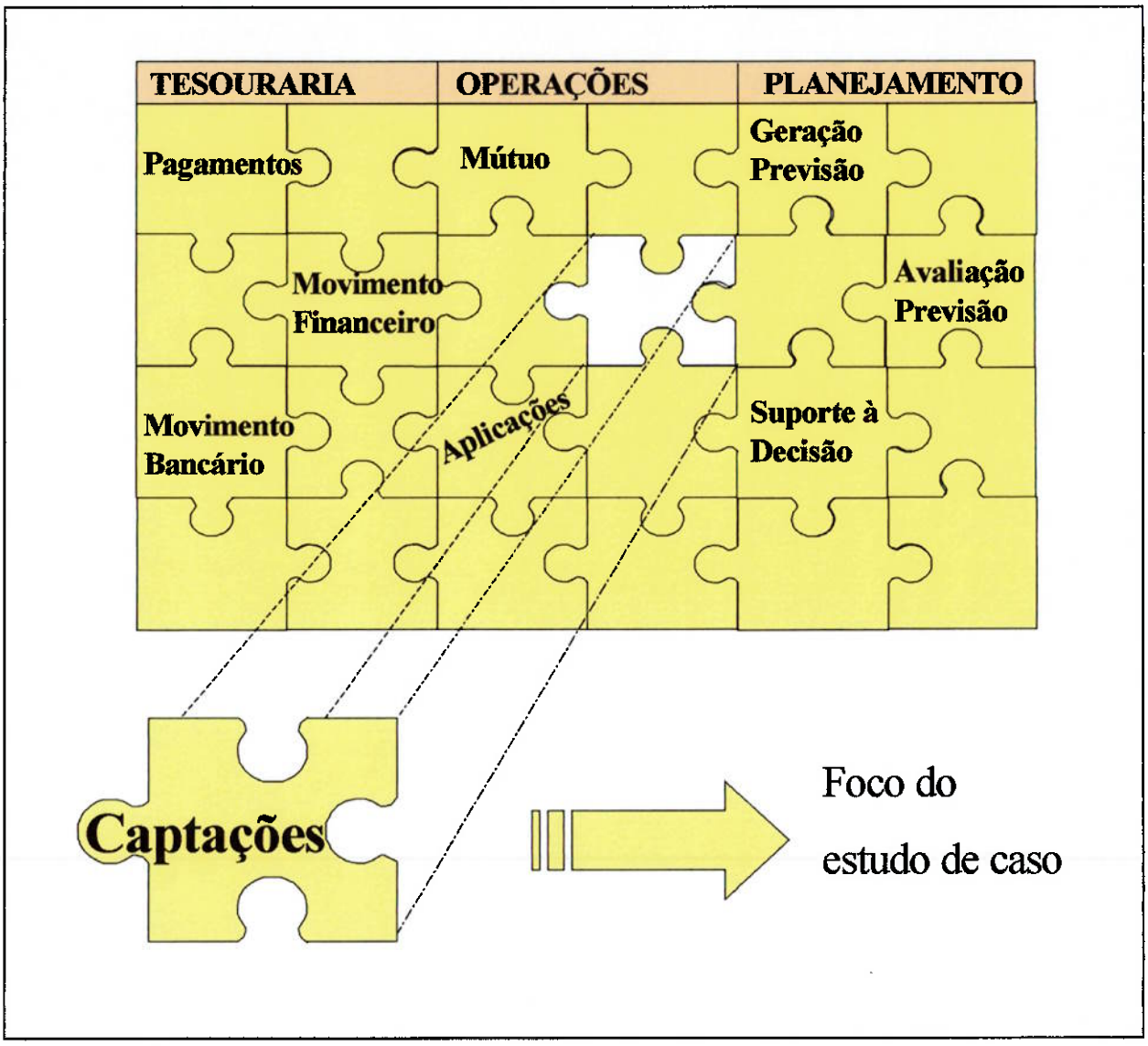


Figura 18 - Escopo do estudo de caso

4.1.2.1 Fluxo de trabalho de Gerenciamento das Captações

Quando surge a necessidade de se contratar um Empréstimo, Financiamento ou Parcelamento, Analistas Financeiros fazem pesquisa com Fornecedores e avaliam as opções de mercado: Modalidade de Captação, Prazo, Taxas.

São realizadas simulações das várias opções em planilha Excel, para comparar os desembolsos que deverão ser efetuados nos diversos Planejamentos; escolhida a melhor opção, é fechado o contrato.

Em seguida, é elaborado o Planejamento Financeiro Previsto, em planilha Excel, apresentado os vencimentos das parcelas de pagamento separadas em componentes na moeda do contrato. Os componentes se dividem basicamente em Amortização de Principal, Juros , Encargos e Correção Monetária. Deve ser feita uma planilha para cada Pagador/Favorecido, de acordo com seu percentual estabelecido.

Conforme as liberações de dinheiro sejam efetuadas, será feito um novo Planejamento Real com a mesma estrutura do Planejamento Previsto, a partir da data da liberação, sendo calculado com o valor da liberação. A liberação de dinheiro é conhecida como Aviso de Crédito.

Repactuações previstas em determinados períodos poderão ocorrer, as quais já foram determinadas em contrato, que consistem em atualização da taxa de juros por um índice pré-fixado, sendo, portanto, recalculados todo os Planejamentos dos Avisos de Crédito.

Poderão ser efetuadas liquidações parciais ou liquidação total dos Contratos, com o recálculo dos Planejamentos dos Avisos de Crédito na data da liquidação.

Para cada vencimento de prestações é feita Autorização de Pagamento e enviado para a área de Contas a Pagar.

Para cada mês são enviados as Provisões para a Contabilidade da seguinte forma:

- Valores provisionados do início do mês até o primeiro vencimento
- Valores provisionados de cada parcela dentro do mês
- Valores provisionados da última parcela até o final do mês

Para melhor entendimento de como são compostas as parcelas de uma operação financeira, e como elas são calculadas, deve ser consultado o anexo C.

4.1.3 Modelo Preliminar de Classes

Com base no conhecimento do negócio, há condições de esboçar as classes representativas do módulo a ser projetado.

Inicialmente, não é necessário se preocupar com as operações e os atributos, porém, já podem ser identificados pelo levantamento do sistema, especialmente os atributos.

Pelo entendimento do negócio e detalhamento do fluxo de trabalho do escopo do projeto, podem ser identificadas, inicialmente, as seguintes classes:

- Operação Financeira
- Índice Econômico
- Modalidade
- Planejamento
- Parcela de Operação Financeira

4.1.4 Casos de Uso

Visando melhor entendimento dos requisitos, devem ser especificados os casos de uso para funcionalidades do sistema, tendo em mente a solução futura.

Foram elaborados os seguintes casos de uso:

1. Determinar tipos de componentes de cálculo
2. Elaborar componentes de cálculo
3. Verificar palavras reservadas
4. Elaborar condições de cálculo
5. Simular captações

As especificações de Caso de Uso encontram-se no Anexo D.

4.1.5 Complemento do Diagrama de Classes

Com base na descrição dos casos de uso, podem ser adicionadas mais classes ao Diagrama de Classes. No exemplo deste Estudo de Caso, foram acrescentadas as seguintes classes:

- Componente de Cálculo
  - Valor Base
  - Valor Derivado
  - Função do Sistema

- Condição de Cálculo

É importante ressaltar que essa classe "Componente de Cálculo" e "Condição de Cálculo" são entidades fortes do negócio que surgiram após pensarmos em uma solução futura para o sistema.

Dificuldades e erros comuns

Serão expostos erros comuns que podem ocorrer ao desenvolver um Diagrama de Classes, especialmente quando se está iniciando na metodologia OO:

1) Incluir entidades de implementação no Diagrama de Classes

Ao se pensar na solução futura, duas entidades haviam sido sugeridas como candidatas a esse primeiro modelo de classes, para assessorar a elaboração do Planejamento de parcelas:

- Modelo: é gerado um modelo para cada operação financeira do sistema indicando um código e freqüência de cálculo.

- Itens de Modelo: apresenta os componentes de cálculo, a ordem de cálculo, a parcela inicial no Planejamento e o número de vezes em que ocorrem no Planejamento.

Exemplo: Modelo 047 – Capital de Giro

Itens de Modelo					
Componente de Cálculo	Ordem Cálculo	Freqüência	Parcela Inicial	Número Vezes	Numero Dias
CTR01	001	MENSAL	0.1	1	0
AMP01	002	SEMESTRAL	2.1	1	0
PRS01	003	SEMESTRAL	2.1	1	0
TXJ01	004	MENSAL	0.1	3	0
JUR02	005	MENSAL	0.1	3	0
JURAC	006	MENSAL	0.1	3	0
PRS04	007	SEMESTRAL	2.1	1	0
SDV01	008	MENSAL	0.1	3	0

Tabela IV - Estudo de caso - exemplo de itens de modelo

Com base nesse modelo de cálculo e considerando uma data-base de, por exemplo: 10-set-2003, seria gerado um Planejamento, conforme indicado na tabela V:

Número Parcela	Componente de Cálculo	Data
0.1	CTR01	10-SET-2003
0.1	TXJ01	10-SET-2003
0.1	JUR01	10-SET-2003
0.1	JURAC	10-SET-2003
1.1	TXJ01	10-OUT-2003
1.1	JUR01	10-OUT-2003
1.1	JURAC	10-OUT-2003
1.1	SDV01	10-OUT-2003
2.1	AMP01	10-NOV-2003
2.1	PRS01	10-NOV-2003
2.1	TXJ01	10-NOV-2003
2.1	JUR02	10-NOV-2003
2.1	JURAC	10-NOV-2003
2.1	PRS04	10-NOV-2003
2.1	SDV01	10-NOV-2003

Tabela V - Estudo de caso - exemplo de planejamento de parcelas

O modelo, além de ser entidade fraca, tem informações já constantes na classe Parcelas, e desta forma poderia criar uma redundância perigosa para o Diagrama de Classes.

Além disso, essa não seria a única solução existente para gerar um planejamento de parcelas, pois outras soluções poderiam ser formuladas, e o diagrama de classes constitui-se de classes estáveis.

2) Redundâncias a serem eliminadas do diagrama de classes

Em um primeiro momento, o seguinte erro poderia ocorrer, ao confundir um Diagrama de Classes com um DER:

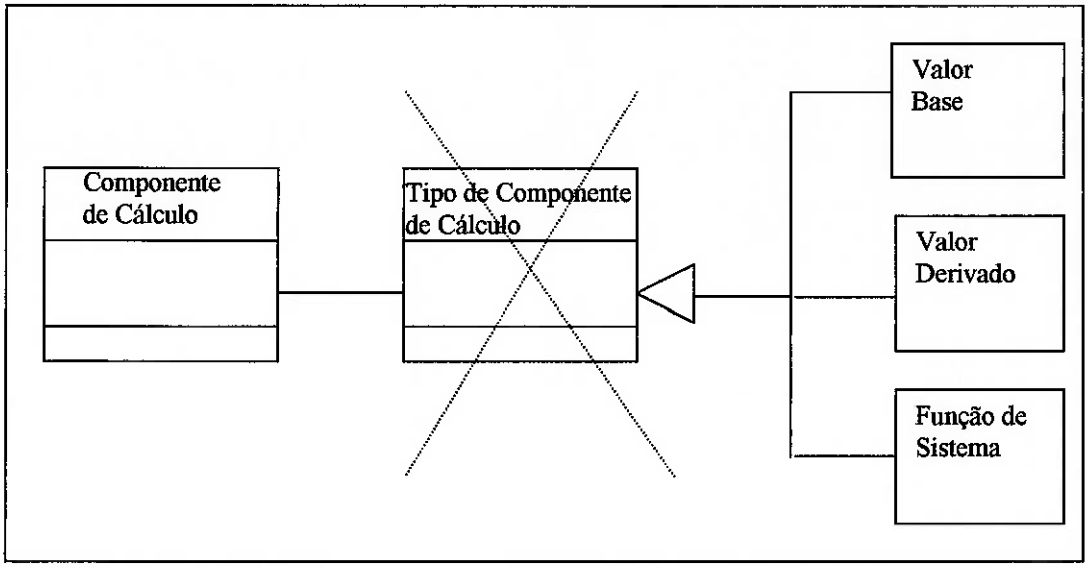


Figura 19 - Redundância em um diagrama de classes

O símbolo indicado na figura 19, que representa a herança, já indica implicitamente o sentido de "Tipo de", ou seja, um Componente de Cálculo pode ser do tipo Valor Base ou pode ser do tipo Valor Derivado, portanto, devemos eliminar a classe Tipo de Componente de Cálculo, a qual constará somente no DER.

É preciso, portanto, além de determinar e organizar as classes de negócio importantes para a modelagem das classes, conhecer o significado das notações da linguagem utilizada, no caso desse Estudo de Caso, a UML, para utilizá-las corretamente.

Poderia ocorrer, também, o seguinte erro, conforme indicado na figura 20, abaixo:

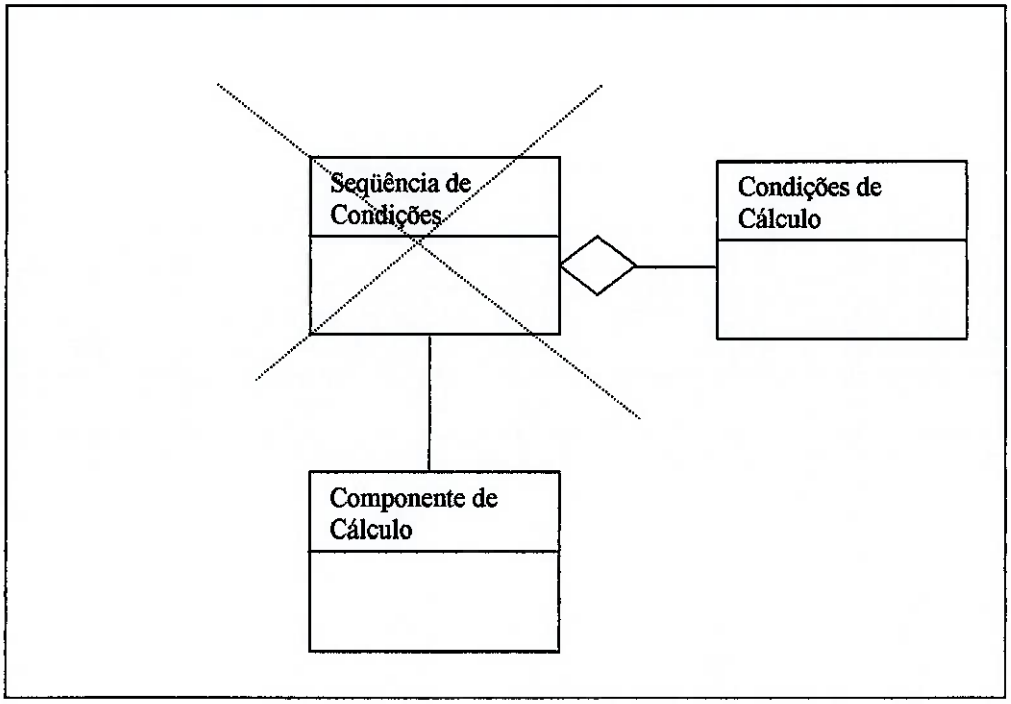


Figura 20 - Redundâncias em diagrama de classes

A classe Seqüência de Condições também está redundante, pois já existe o símbolo de agregação que já indica que há uma seqüência de condições que podem existir para um componente de cálculo; portanto, deve ser eliminada.

Não se deve pensar em detalhes de implementação para desenvolver o diagrama de Classes.

3) Identificar as sub-classes de uma herança independente da forma como forem incrementadas ao sistema.

   Não havia sido representadas as seguintes sub-classes da classe modalidade:

- Finame
- Eletrobrás
- ICMS
- BNDES

Já as sub-classes da classe Componente de Cálculo foram desde o início representadas:

- Valor Base
- Valor Derivado
- Função de Sistema

A diferença entre esses dois grupos de sub-classes é a seguinte:

As sub-classes da classe Componente de Cálculo surgirão da construção do sistema, caso aparece um novo tipo, deverá ser implementado no sistema através de manutenção em código.

As sub-classes da classe Modalidade poderão ser cadastradas pelo usuário.

Não importa a forma como serão incrementadas novas sub-classes, o importante é que ambas sejam representadas no diagrama de classes.

4.1.6 Diagrama de Seqüência

O diagrama de seqüência deste estudo de caso foi elaborado considerando diversos cenários, por serem situações independentes uma das outras:

- Inclusão de novos períodos na simulação da operação financeira
- Inclusão de novos componentes de cálculo
- Cálculo da operação financeira

O diagrama de seqüência está representado no Anexo F.

4.1.7 Complemento do Diagrama de Classes

Com base na descrição dos diagramas de seqüencia, pudemos complementar o Diagrama de Classes atribuindo as responsabilidades pelas operações às classes do sistema.

O diagrama de classes completo está no Anexo E.

4.2 Especificação da Arquitetura

Neste estudo de caso, até então nos preocupamos com o domínio do negócio em estudo, mas sabemos que um sistema não se constitui apenas desses objetos. Um sistema é composto de vários subsistemas dos quais os objetos do domínio do problema são apenas um deles. Geralmente um sistema de informação deve estar conectado a uma interface de usuário e a um mecanismo de armazenamento persistente.

Se fôssemos considerar um sistema para WEB, o ambiente em que o sistema se instalaria seria ainda mais complicado, composto de diversos containeres cada qual desempenhando seu papel específico, como por exemplo: servidor de HTTP, servidor que contemplam serviços para páginas dinâmicas, etc.

Esse ambiente complexo que deve ser estruturado em paralelo ao projeto do sistema é a arquitetura.

Para esse estudo de caso, será adotada a arquitetura clássica de três camadas, que é uma arquitetura comum para sistemas de informações que incluem um front-end de comunicação com usuário e o armazenamento persistente de dados.

Basicamente cada camada se constitui dos seguintes elementos:

1. Apresentação : janelas, relatórios
2. Lógica da Aplicação : regras do domínio do problema
3. Armazenamento : Banco de Dados

As vantagens para a adoção desse tipo de arquitetura envolvem:

- Com o isolamento da lógica de negócios em componentes separados, é possível sua reutilização em outros sistemas.

- Aumento de produtividade pelo desenvolvimento em paralelo, suportado pela alocação de desenvolvedores para a construção de camadas específicas, por exemplo Ter uma equipe trabalhando exclusivamente na camada de apresentação.

- Distribuição de camadas em diferentes nós físicos de processamento pode melhorar o desempenho e aumentar a coordenação e o compartilhamento de informações em um sistema cliente-servidor.

Neste estudo de caso, será escolhida a arquitetura multi-camada J2EE que é um padrão de desenvolvimento em camadas que disponibiliza uma série de serviços de infraestrutura de alto-nível evitando o desenvolvimento de código complexo, aproximando os desenvolvedores do negócio.

J2EE é tipicamente consistido das seguintes camadas:

Camada Client: Browser/HTML, Java Applets, Java Applications, Flash e outros.

Camada WEB: é responsável por comunicar-se com a camada de negócio, possui lógica de apresentação e recepção de dado, tipicamente gerar HTML/XML. São implementados no J2EE com Servlets e JSp.

Camada de negócio: é a inteligência da aplicação. Na maioria das ocasiões é implementado com a API/Framework Enterprise JavaBean. O Application Server ou o container EJB vai prover serviços de persistência, transação, alocação de recursos e gerenciamento de ciclo de vida de componente.

O J2EE é uma boa escolha por ser:

- Integrável
- Extensível
- Modular
- Seguro

- Disponível e escalável

A seguir uma representação esquemática extraída do site da GlobalCode sobre a arquitetura J2EE:

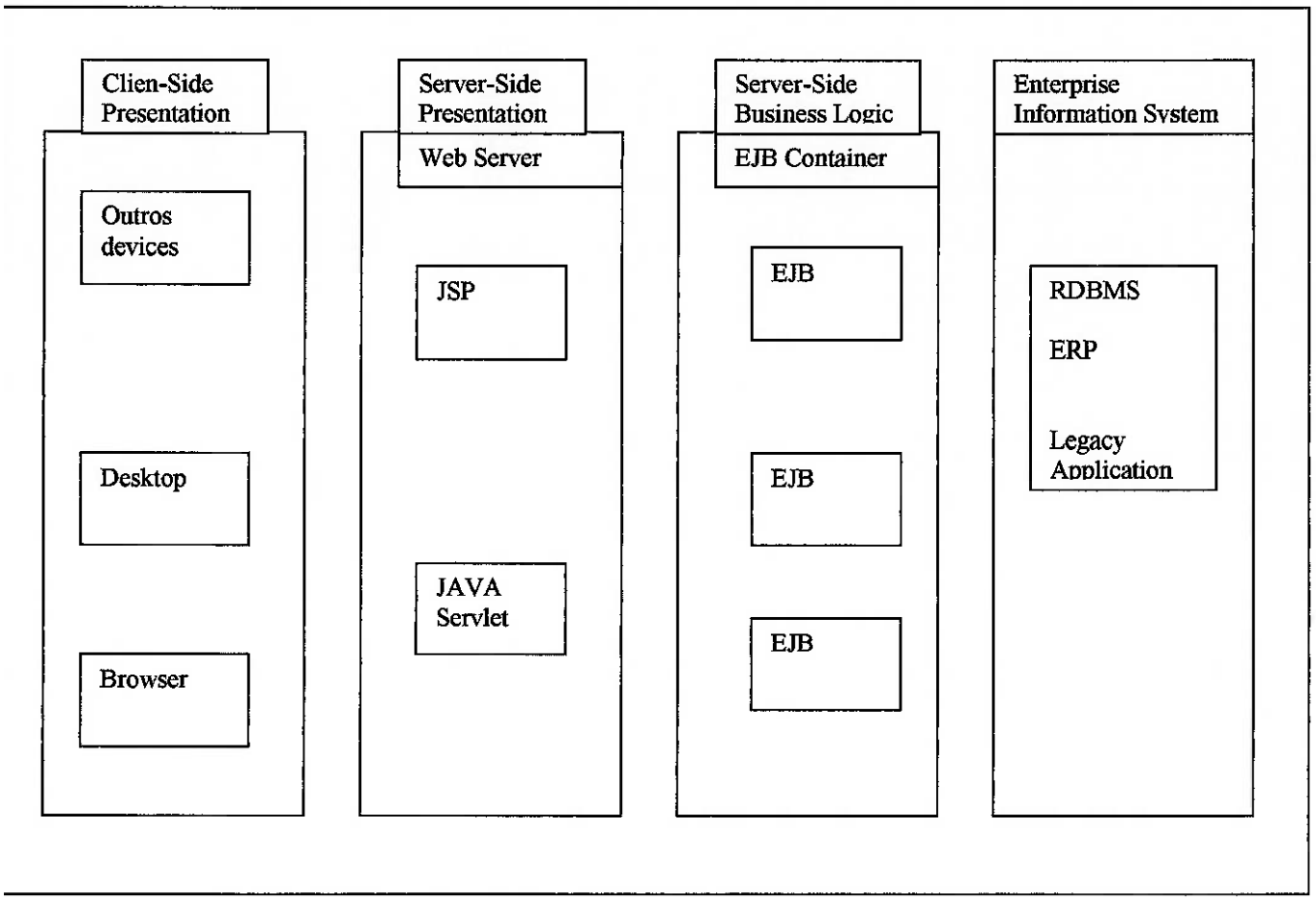


Figura 21 - Representação da arquitetura J2EE

A camada de dados ou Enterprise Integration System (EIS): é a camada responsável pelos dados e objetos persistentes. Classes de acesso a banco de dados, conectores para ERP e funcionalidades personalizadas de I/O compõem essa camada.

Por essa arquitetura, pode-se especificar aplicações com thin-client e fat-client, coexistindo em um mesmo ambiente.

Um thin-client pode ser acessado diretamente com um applet o servidor WEB.

Um fat-client pode acessar via XML o servidor WEB, caso esteja sendo acessado via Internet. Caso esteja sendo acessado pela Intranet, não há essa necessidade.

4.3 FASE 2: Projeto

4.3.1 DER

Agora pode-se ter uma clara conclusão sobre as diferenças entre o Diagrama de Classes e o DER. Para exemplificar vamos comparar uma pequena parte do Diagrama de Classes com a parte correspondente no DER:

Exemplo de parte do Diagrama de Classes:

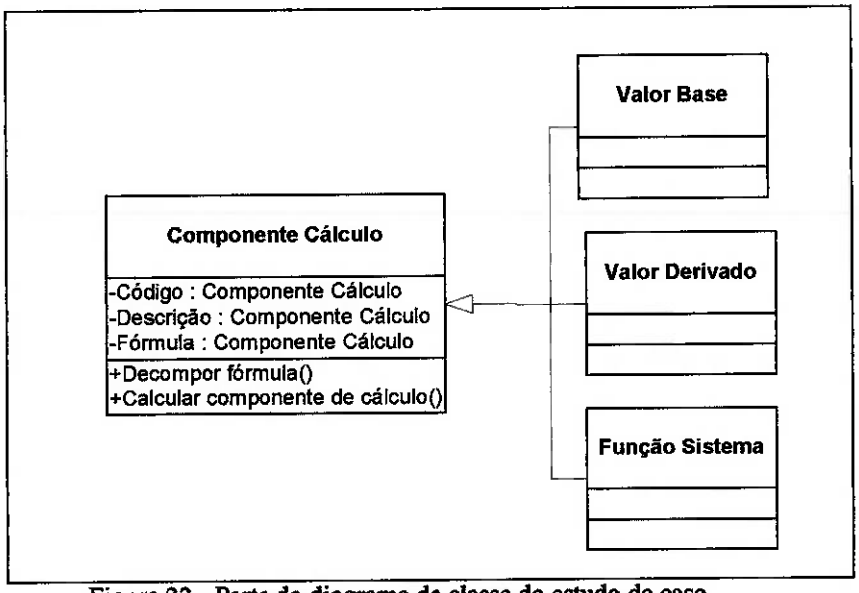


Figura 22 - Parte do diagrama de classe do estudo de caso

Parte do DER correspondente ao exemplo anterior :

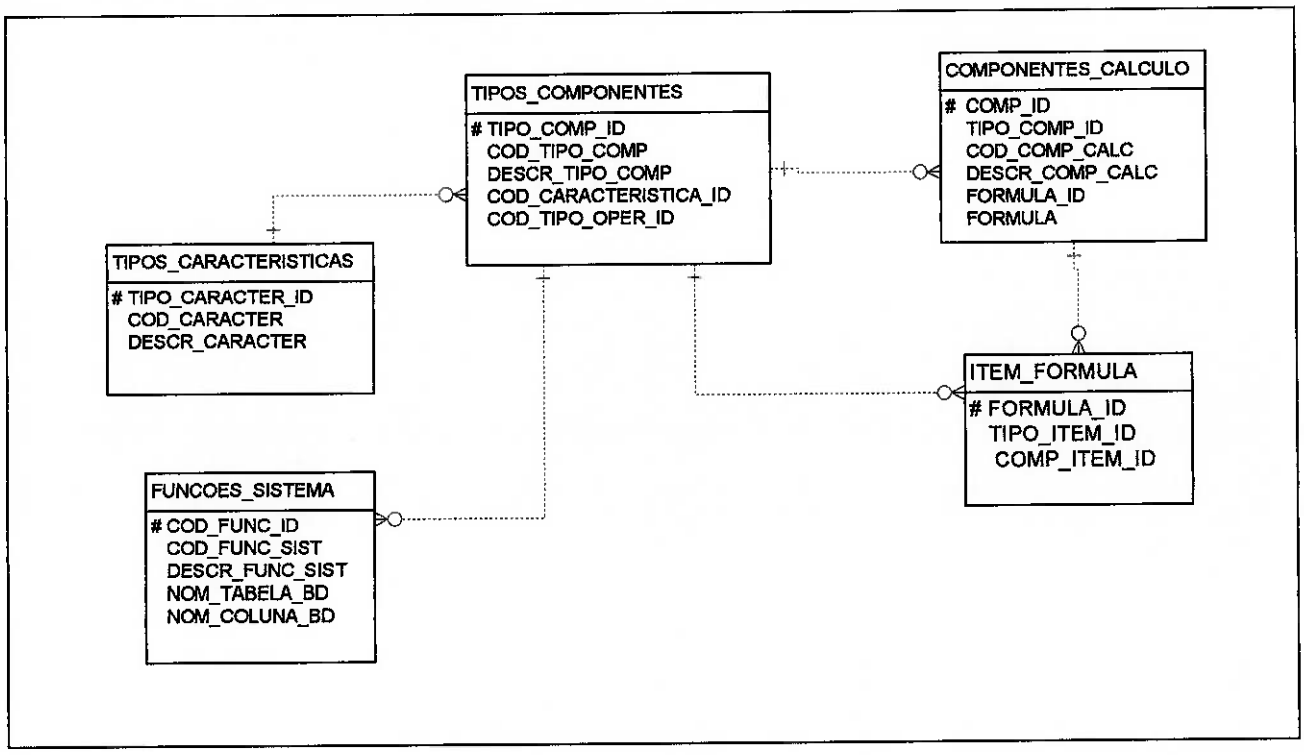


Figura 23 - Parte do DER do estudo de caso

Perceba que certos detalhes de implementação do DER não são classes. As entidades fracas, por exemplo, ITEM_FORMULA, TIPOS_CARACTERISTICAS não correspondem a classes do Diagrama de Classes.

Para fazer o mapeamento do Diagrama de Classes para o DER, utilizamos conceitos propostos no item "Mapeamento de classes persistentes para o DER" do anexo A.

Para a herança indicada pela superclasse Componente e as subclasses Valor Base, Valor Derivado e Função de Sistema, foi utilizado em parte o recurso de Fragmentação Vertical pois praticamente criamos uma tabela para cada classe:

A superclasse Componente está representada pela tabela COMPONENTES_CALC, a subclasse Função de Sistema está representada pela tabela FUNCOES_SISTEMA; já as subclasses Valor Base e Valor Derivado, por

serem muito semelhantes e apresentarem praticamente os mesmo atributos, estão apenas diferenciadas pelo tipo, elas estão representadas pela tabela TIPOS_COMPONENTES e se diferenciam pelo campo COD_CARACTERISTICA_ID.

O DER completo desse estudo de caso encontra-se no anexo H.

4.3.2 Diagrama de Componentes

Neste estudo de caso, estabelecemos a partir do diagrama de classes, 4 componentes com classes afins.

Componente 1

É o componente principal do sistema, responsável por direcionar a simulação de operações financeiras e apresentá-las ao usuário.

Compõe-se pelas classes: Operação Financeira e Parcela.

Componente 2

É o componente de cálculo.

Compõe-se pelas classes: Componente de Cálculo e Condição de Cálculo.

Perceba que pela modularidade estabelecida, posso efetuar um cálculo dentro de um operação financeira ou de um componente em particular.

Componente 3

É o componente de Modalidades.

Compõe-se pela classe Modalidade.

Componente 4

É o componente de Indicadores Financeiros.

Compõe-se pelas classes: Cotação e Indicador. Responsável por determinar as cotações de moedas e indicadores em períodos especificados, calcular cotações entre indicadores diferentes, estabelecendo a conversão entre eles.

Perceba que para a criação de cada componente, reunimos as classes com responsabilidades afins. Por exemplo, para um componente de cálculo, precisamos tanto da classe componente de cálculo como da classe de condição de cálculo, uma complementa a outra para prestar o serviço de cálculo de componentes.

No anexo I está o particionamento que fizemos no diagrama de classes para determinar os componentes. Em seguida está a representação de cada componente em UML.

Para cada componente, estabelecemos os serviços prestados pelo componente, quanto os serviços requisitados pelo componente do ambiente em que ele atuará.

Como a arquitetura escolhida foi JAVA, estabelecemos os serviços que cada componente precisará:

- Conexão com o Banco de Dados : JDBC
- Bibliotecas de classe do JAVA : JDK
- Run-time do JAVA: JRE
- Serviço para montar páginas HTML dinâmicas: JSP

4.3.3 IHM

O anexo J apresenta os requisitos da Interface Homem-Máquina, definidos para atender aos diversos perfis de usuário.

Em seguida, são esboçadas as telas das principais funcionalidades do sistema, para que facilite a visualização de seu funcionamento.

5 CONSIDERAÇÕES FINAIS

Neste capítulo final apresenta-se as principais conclusões encontradas neste trabalho, assim como, serão apresentados os comentários gerais, sugestões para impulsionar próximos trabalhos e a contribuição acadêmica.

5.1 Conclusões

É importante que esse trabalho auxilie muitos profissionais de TI que são formados nas metodologias tradicionais que ainda não compreenderam o significado da Orientação a Objetos e sua contribuição, que acreditem ser uma técnica exageradamente complexa e completamente diferente das técnicas tradicionais que vem adotando, desmistifiquem essa idéia.

Descrição de Requisitos é o item mais importante de um sistema, que a UML fornece ferramentas para muito bem defini-los e descrevê-los; mas não são mágicos, depende muito da habilidade do analista em capturá-los. A técnica dos casos de uso dá um bom direcionamento pelas regras de composição do documento/diagrama, mas o conteúdo parte do bom levantamento do analista.

Reusabilidade, por exemplo, que é uma das grandes promessas da tecnologia orientada a objetos, não pode ser obtida apenas utilizando as ferramentas de desenvolvimento orientado a objetos; é necessário um grande exercício prévio de abstração.

Essas características de bom levantamento, abstração do problema são inerentes a um bom profissional em desenvolvimento de sistemas, independente da

técnica que esteja adotando: Análise Estruturada ou Orientação a Objetos; o que ocorre é que a OO pode dar mais suporte, mas a matéria-prima que é a informação obtida deve vir do profissional que elabora os produtos da técnica.

Mas é claro que a OO ajuda muito pelos recursos que possui tanto em modelagem como em implementação. A UML traz artefatos que se complementam e permitem um desenvolvimento incremental. A programação em OO apresenta o tripé Encapsulamento, Herança e Polimorfismo com recursos que induzem às melhores técnicas.

Vale ressaltar que a Orientação a Objetos não é apenas uma onda do momento, é uma necessidade, dado a atual estágio de desenvolvimento do software.

O desenvolvimento voltado para a WEB, a integração de sistemas, o desenvolvimento em várias camadas, praticamente obrigam que seja utilizada a técnica OO.

Um profissional metódico e organizado consegue obter boas arquiteturas mesmo utilizando análise estruturada, utilizando implicitamente conceitos de reutilização de código, modularização e encapsulamento. Para quem já faz isso, migrar para OO será fácil.

O importante é que o profissional que queira se atualizar não encare a OO como um "bicho de 7 cabeças". Há menções a um artigo muito interessante no lista de referências que trata de uma experiência bastante interessante ocorrida em um projeto importante de TI na HP que foi bem sucedido apesar de contar com profissionais que não tinham experiência em OO , apenas em Análise Estruturada, a conduzir um projeto utilizando a técnica OO. Não só os profissionais envolvidos diretamente no projeto, mas como toda a corporação não tinha a cultura de Orientação a Objetos. Esse artigo deve servir como um bom motivador para que os profissionais se arrisquem nessa empreitada.

5.2 Continuidade do trabalho

É importante reconhecer que esse trabalho é um estágio inicial dentro de um longo estudo que deve permear a pesquisa dentro da tecnologia de Orientação a Objetos.

Existem muitos conceitos, métodos, técnicas e ferramentas da Engenharia de Software para desenvolvimento de *software*, que devem ser exploradas, conforme sugere-se nos itens abaixo:

- UML: durante todo o trabalho desta monografia, ilustramos os conceitos de Orientação a Objetos, assim como fizemos os diagramas para modelagem do projeto do Estudo de Caso em UML. Sugere-se o aprofundamento no estudo desta linguagem, por ser um padrão amplamente adotado atualmente para modelagem de *software* orientado a objeto, e também por estar evoluindo, brevemente será feito o lançamento da nova versão da UML 3.0, em que novos diagramas, além dos 9 (nove) diagramas existentes hoje, estarão sendo lançados, visando aumentar a integração de especificações de sistema, permitindo a geração de código e engenharia reversa, talvez até levando a ponto de executar alguns modelos UML.

- *Design Patterns:* são padrões aceitos como sendo as melhores práticas para solução de problemas conhecidos ocorridos no desenvolvimento de software. É a introdução de catálogos de soluções, já existente em outras ciências, à área de Engenharia de *Software*. A representação dos *Design Patterns* pode ser observada e explicada em diagramas UML, especialmente através de diagramas de classe, o que demonstra a sua relação próxima com a Orientação a Objetos.

- Objetos Distribuídos: conforme cita Becerra em sua tese sobre ODP, "na área de orientação a objetos aumentar-se-á as pesquisas com o objetivo de equacionar os benefícios da orientação a objetos aplicados a sistemas distribuídos. Estes objetivos serão atingidos realizando projetos distribuídos baseados em OO". Portanto, é também uma grande área de interesse, dando continuidade ao trabalho de pesquisa dentro da Tecnologia de Orientação a Objetos. Grandes temas de destaque nessa área podem ser explorados: CORBA, WEB-SERVICES, MOBILE COMPUTING e DCOM.

- Tecnologia JAVA: Java é uma tecnologia totalmente orientada a objetos. É um grande tópico de estudo, especialmente para ser aplicado durante a fase de implementação do desenvolvimento de *software*. É uma tecnologia que está em destaque hoje, e acredita-se que seja estável e duradoura, não meramente um modismo, por ser robusta, portável e flexível.

- XML: É a tecnologia que pode ser combinada ao JAVA, fazendo a dupla perfeita entre código e dados portáveis. XML trata-se de uma especificação padrão para identificação e descrição de dados.Muito se tem evoluído em termos de interoperabilidade entre sistemas graças ao auxílio da XML.

5.3 Contribuição Acadêmica

A contribuição acadêmica define a forma como este trabalho de monografia aumentou as fronteiras do conhecimento e aplicabilidade da Orientação a Objetos, especialmente para profissionais experientes em Análise Estruturada.

O ponto mais importante da contribuição acadêmica, está na aplicabilidade da proposta de migração da metodologia de Análise Estruturada para a Orientação a Objetos, esta proposta deverá contribuir para a aceitação ampla desta tecnologia, especialmente dentre o público alvo deste trabalho.

É importante frisar que existe ainda muitos aspectos para pesquisar na tecnologia de Orientação a Objetos.

Finalmente, este trabalho deve contribuir e impulsionar trabalhos posteriores que possam dar continuidade ao desenvolvimento de pesquisas nessa tecnologia tão excitante que é a Orientação a Objetos.

ANEXO A – MAPEANDO CLASSES PERSISTENTES PARA O DER

Segundo Zimbrão(2003), os principais conceitos de orientação a objetos que aparecem em um modelo de classes e devem ser mapeados para um modelo relacional são: identidade, classes, atributos, métodos, herança, relações (associação, agregação e composição) e restrições.

No geral, poderiam ser estabelecidas as seguintes regras de mapeamento:

Construção	Mapeamento
Identidade	Chaves primárias artificiais
Classes	Tabelas
Atributos	Colunas de tabelas
Métodos	*Stored procedures* ou funções fora do banco de dados
Relações	Chaves estrangeiras e tabelas de relacionamento
Herança	Tabelas, visões e particionamentos horizontais ou verticais
Restrições	*Constraints* e *triggers*

Tabela VI - Mapeamento de classes persistentes para modelo relacional

1. Identidade

   A recomendação dada é que toda tabela que corresponder a uma classe no modelo original possuirá um identificador inteiro, seqüencial e que nunca será reutilizado, como chave primária. É possível haver outras chaves candidatas (alternativas,secundárias), mas a chave primária será sempre este identificador, geralmente, representado como ID. Apenas esse atributo será utilizado para a criação de chaves estrangeiras e relacionamentos entre as tabelas.

2. Classes

   De forma geral, cada classe simples será mapeada para um tabela. Classe simples são classes cujos atributos podem ser mapeados

diretamente para os tipos básicos do banco de dados, além de não fazer parte de nenhuma hierarquia de classes.

3. Atributos

As quatro características dos atributos são:

3.1 Tipo

O modelo de classes permite que um objeto tenha atributos de tipos complexos, incluindo tipos abstratos de dados ou até mesmo outros objetos. Mapear estes atributos para um modelos relativamente simples como o relacional deve ser feito de forma criteriosa.

3.2 Cardinalidade

Deve-se prestar atenção quando aparecerem coleções. Uma forma de tratar é criar uma tabela auxiliar contendo uma chave estrangeira para o ID da tabela original. Se a coleção não permitir repetição, deve-se criar uma restrição de unicidade contendo o campo e a chave estrangeira.

Exemplo de classe:

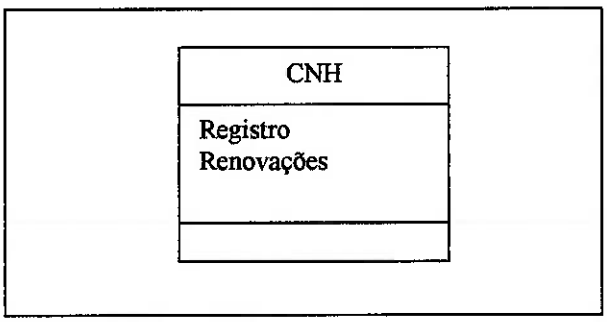


Figura 24 - Classe representada em UML

Exemplo de mapeamento :

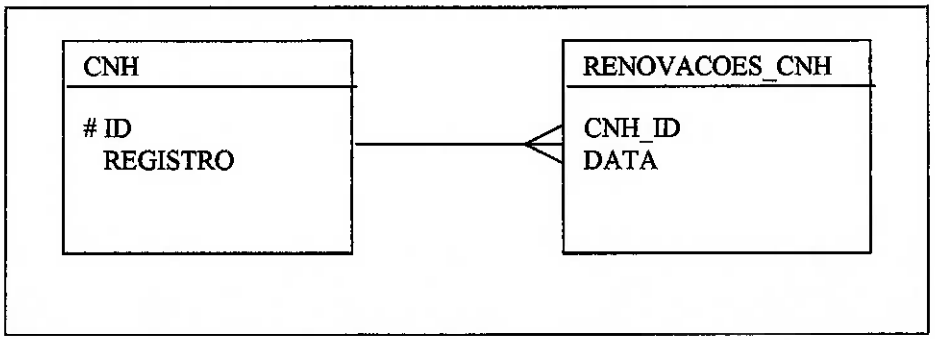


Figura 25 - Exemplo de mapeamento de classes persistentes para um DER

3.3 Persistência

Recomenda-se o uso de visões para atributos não persistentes.

3.4 Visibilidade

Recomenda-se o uso de visões para restringir o acesso a campos e *stored procedures* para realizar as alterações.

4. Métodos

São as funções e procedimentos que poderão ser executados no ambiente do banco de dados ou não.

5. Relações

Basicamente será utilizada a mesma abordagem do modelo relacional, atentar para que as tabelas que representam classes tenham como chave primária apenas o campo ID. Como conseqüência, todas as chaves estrangeiras terão apenas um campo. As cardinalidades devem ser refletidas no mapeamento de acordo com as multiplicidades expostas no Diagrama de Classes.

6. Herança

Segundo Zimbrão(2003), o mapeamento de herança é o que apresenta maior dificuldade e também o que traz mais conseqüências, tanto no desempenho quanto na facilidade de escrita de consultas. As escolhas são:

6.1 Todas as classes da hierarquia em uma única tabela

Neste caso representar todas as classes de uma hierarquia em uma única tabela, acrescentando um atributo TIPO para identificar a classe do objeto.

A vantagem é a recuperação fácil das informações.Como desvantagem há disperdício de espaço.

6.2 Fragmentação vertical

No exemplo:

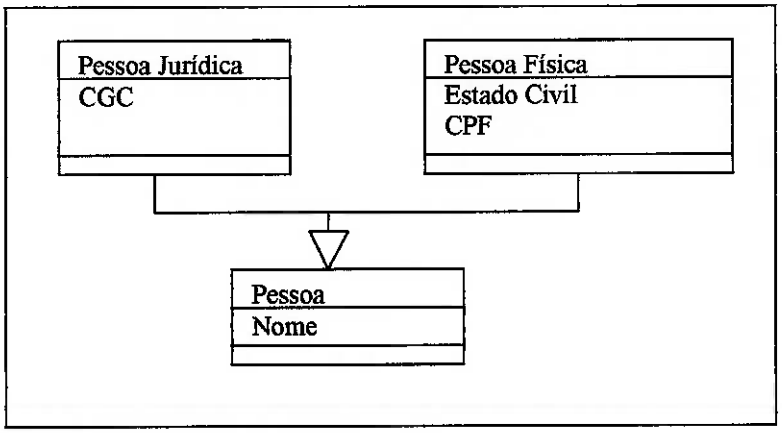


Figura 26 - Herança representada em UML

Neste caso seriam criadas 3 tabelas, representando os atributos comuns apenas na tabela que representa a classe pais.

6.3 Fragmentação horizontal

Neste caso seriam criadas 2 tabelas, representadas pelas classes concretas da hierarquia contendo todos os seus atributos e os de suas superclasses, ou seja, uma tabela representando a pessoa jurídica e outra representando a pessoa física.

7. Restrições

Podem ser controladas por mecanismos do próprio banco de dados ou por triggers.

Zimbrão(2003) faz uma importante menção : "... apesar de se estar usando um modelo OO, todos os mecanismos disponíveis no banco de dados relacional, tais como integridade referencial, unicidade, restrições e validações, devem ser utilizados para garantir o estado válido dos dados. Por outro lado, o que deve guiar um bom mapeamento é o grau de fidelidade ao modelo de classes e objetos, sempre contraposto à razão custo/benefício. Não se deve realizar o mapeamento com um 'pensamento relacional', aplicando técnicas de identificação de entidades e normalização de forma cega – o produto final pode ficar tão distante do modelo original que a programação e manutenção se tornarão extremamente difíceis."

ANEXO B – ARTEFATOS DA ANÁLISE ESTRUTURADA E DA OO

A seguir estão o DFD, o diagrama de classes e o DER do exemplo apresentado no capítulo 3, para que possam ser comparados.

Observando o DFD e o diagrama de classes, percebe-se a inversão que ocorre em OO, no qual as estruturas de dados tomam o lugar dos processos em grau de importância.

Observando o diagrama de classes e o DER, entende-se a diferença entre os dois, pois uma classe persistente pode ser implementada por várias tabelas de um banco de dados relacional. Essa comparação serve para desmistificar a freqüente confusão que se faz entre os dois diagramas.

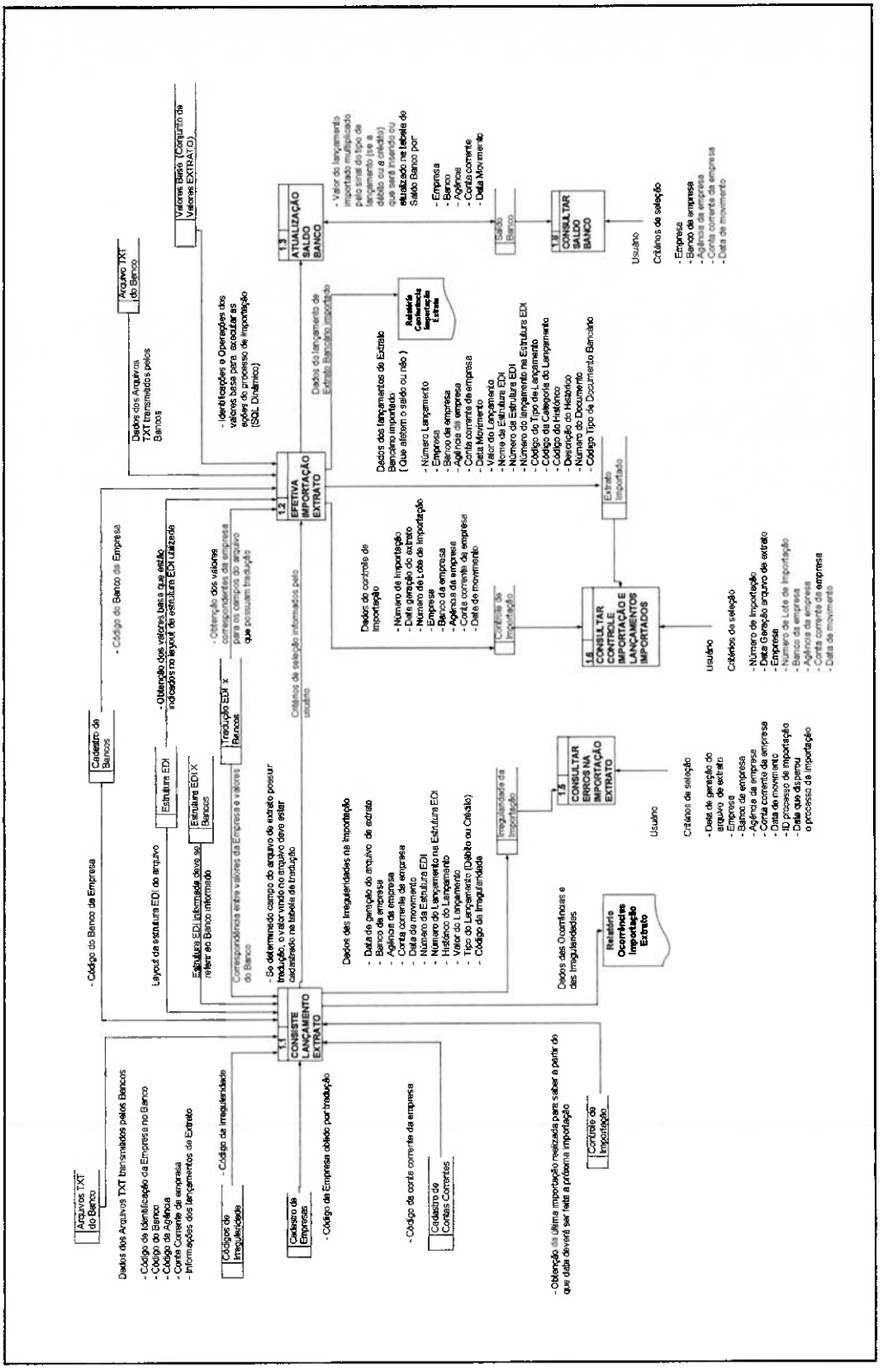


Figura 27 - Exemplo de DFD

A seguir, um exemplo de Diagrama de Classes referente ao mesmo domínio de problema do DFD do exemplo anterior:

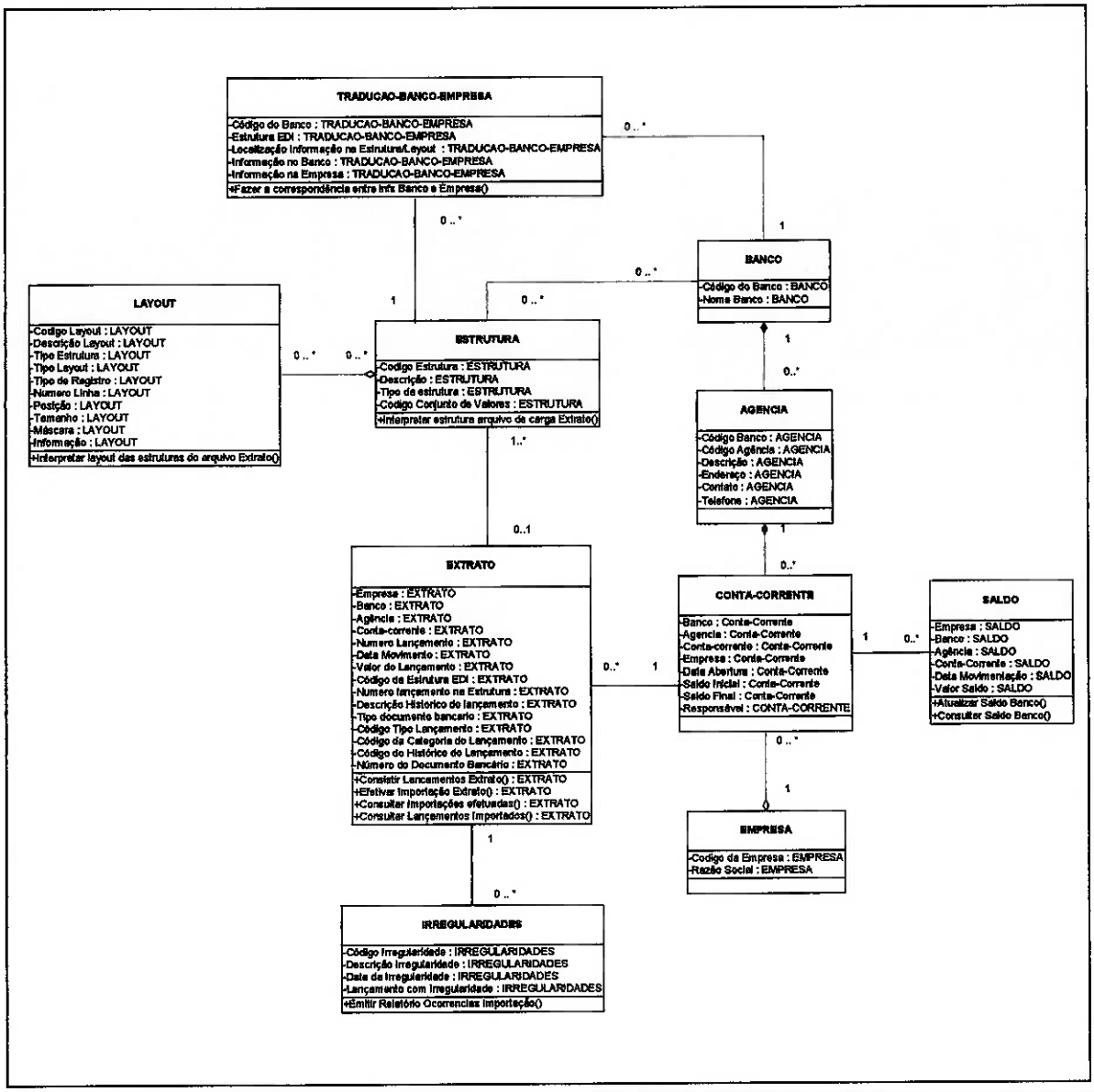


Figura 28 - Exemplo de Diagrama de Classe

A seguir, um exemplo de Diagrama Entidade-Relacionamento referente ao mesmo domínio de problema do DFD do exemplo anterior:

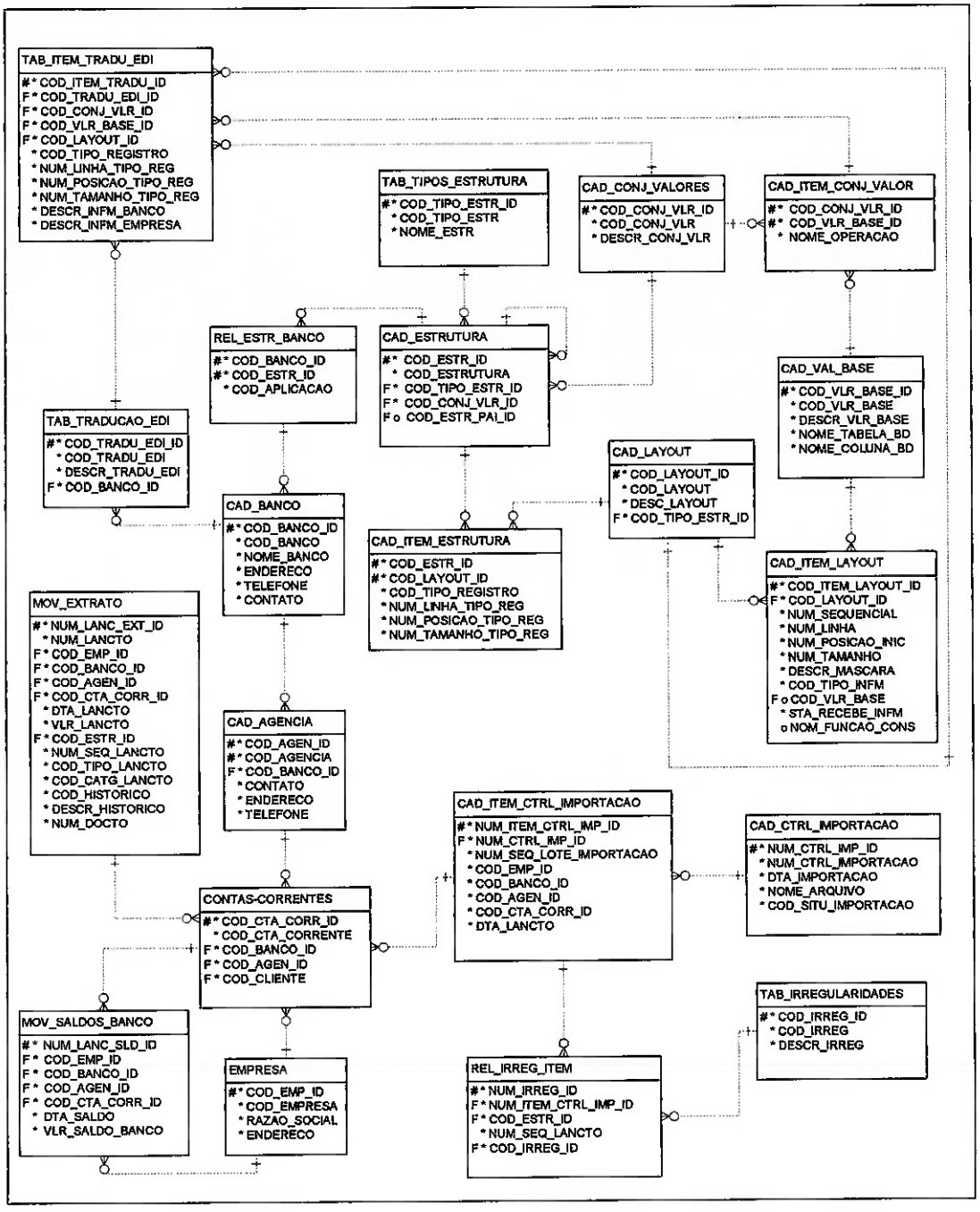


Figura 29 - Exemplo de DER

ANEXO C - REGRAS DE NEGÓCIO

Para melhor compreensão do Estudo de Caso, será exposto, a seguir, conceitos e regras do negócio.

1. Conceitos de Operações Financeiras

1.1 Tipos de Operações Financeiras

Existem basicamente 3 (três) tipos de operações financeiras:

1.1.1 Aplicações

São operações financeiras nas quais há saída de dinheiro, com a condição de que este dinheiro retorne à empresa, juntamente com os rendimentos que ocasionar, ao final de um período estipulado (período da operação). O recebimento de uma aplicação é denominado de resgate.

1.1.2 Captações

São operações financeiras nas quais há entrada de dinheiro na empresa por um determinado período estipulado (período da operação). Ao fim deste período, o dinheiro é devolvido à fonte de origem, juntamente com os rendimentos que ocorrerem durante a duração da operação. O pagamento de uma captação é denominado de liquidação.

1.1.3 Mútuo

São operações realizadas entre duas empresas do mesmo grupo. Nessa operação há uma saída de dinheiro de uma empresa e uma entrada em outra, ou seja, uma das empresas faz o papel de banco. Em outras palavras, uma empresa faz a

aplicação (a que envia o dinheiro) e outra faz a captação (a que recebe), ambas transacionando sem o intermédio de um banco.

1.2 Diferenças entre Aplicações e Captações

Ao contrário do que se pode imaginar, quase não há diferença entre aplicações e captações. Em última análise, todas as operações se resumem a:

Aplicação	Captação
1. Saída de dinheiro	1. Entrada de dinheiro
2. Remuneração dessa saída (Juros + correção monetária)	2. Remuneração dessa entrada (Juros + correção monetária)
3. Recebimento da saída acrescida da remuneração	3. Pagamento da entrada acrescida da remuneração
4. Pagamento dos encargos	4. Pagamento dos encargos

Tabela VII - Diferenças entre aplicações e captações

De acordo com o esquema acima da tabela VII, verifica-se que as variáveis envolvidas tanto em uma aplicação quanto em uma captação são as mesmas: uma parcela de principal; uma parcela de juros; uma parcela de correção monetária e uma parcela de encargos financeiros.

Principal de uma operação financeira é o montante aplicado/captado. Assim, se foi feita uma aplicação de $ 1.000.000,00; esse é o valor do principal da aplicação.

Observa-se, portanto, que a única diferença entre uma aplicação e uma captação é o sentido das parcelas, conforme indicado abaixo, na tabela VII.

	Aplicação	*Captação*
Principal	Saída (você aplica o dinheiro)	Entrada (você recebe o dinheiro emprestado)
Principal + Juros + Correção monetária	Entrada (o banco lhe pagará)	Saída (você pagará ao banco)
Encargos	Saída (você pagará ao Estado/Banco)	Saída (você pagará ao banco)

Tabela VIII - Diferenças entre aplicações e captações

1.3 Os Componentes de uma Operação

1.3.1 Principal, Juros, Correção Monetária e Encargos

Uma operação financeira tem 4 tipos de componentes: de principal, de juros, de correção monetária e de encargos financeiros.

1.3.1.1 A Parcela de Principal

A parcela de principal é o montante aplicado/captado.

Para uma aplicação sem resgates ou depósitos intermediários, ela é constante no tempo, isto é, não tem correção alguma. Por exemplo, se no dia 07/07 forem aplicados $1.000.000,00 a parcela de principal terá exatamente este valor. Um mês depois, ela (a parcela) ainda terá o mesmo valor ($1.000.000,00).

Se houver resgates parciais ou depósitos intermediários, esta parcela pode sofrer alterações. No exemplo anterior, se no dia 15/07 forem aplicados mais $500.000,00 na mesma conta de aplicação, então, a partir daí a parcela de principal valerá $1.500.000,00.

1.3.1.2 Forma de Valorização Total da Operação

A variável mais importante de uma operação financeira, seja aplicação ou captação, é a forma de remuneração de seu principal. Esta remuneração pode ser de três formas:

- **Pré-fixada**: a remuneração do principal será calculada a partir de uma taxa previamente negociada com o agente financeiro (banco);
- **Pós-fixada**: a remuneração do principal será calculada a partir da variação de um indexador. Por exemplo, os fundos de *commodities* são calculados a partir da variação das cotas de seus respectivos bancos;
- **Mista**: a remuneração do principal será calculada a partir de uma taxa mais a variação de um indexador : um exemplo deste tipo de

valorização é um CDB remunerado a 99,6% do CDI mais 6,00% ao ano.

Existem operações que a princípio podem parecer pós-fixadas, mas na verdade não o são. Um exemplo atual é a caderneta de poupança, que tem sua valorização calculada pela TRD do dia da aplicação. Ora, desta forma a taxa de valorização já é conhecida no momento do fechamento do negócio, sendo portanto uma operação pré-fixada. Operações pós-fixadas têm sua valorização conhecida apenas no dia de seu vencimento.

Além disso, para operações pós-fixadas ou mistas, muitas operações não usam a variação completa do índice em um período, mas apenas uma parte dessa variação, chamada pelo sistema de percentual do índice de valorização. Outra informação importante é determinar se este percentual é aplicado sobre a variação completa do índice no período, ou se é aplicada sobre a variação de cada dia do indexador.

Exemplo: Suponha uma aplicação na modalidade "CDB Pós-Fixado" valorizada (índice de valorização) a 90% do indexador CDI (Certificado de Depósitos Interbancários). A seguir, será apresentado como fica o saldo desta operação após três dias úteis, nas seguintes situações:

Situação 1: 90% for aplicado sobre a variação de cada dia (Dia-Dia);

Situação 2: 90% for aplicado sobre a variação acumulada (Acumulado).

Valores de CDI para três dias úteis :

Data	Valor do CDI
08/08	10 %
09/08	12 %
10/08	10%

Tabela IX - Valores de CDI

Situação 1. Dia-Dia

Valores de CDI e acumulados para três dias úteis :

Data	Valor do CDI	Fator de acumulação = (valor * 0.90)/100 + 1
08/08	10 %	10 * 0.90 / 100 + 1 = 1.090
09/08	12 %	12 * 0.90 / 100 + 1 = 1.108
10/08	10 %	10 * 0.90 / 100 + 1 = 1.090

Tabela X - Valores de CDI e acumulados na situação 1

O acumulado para os três dias neste caso é : 1,090*1,108*1,090 = 1,3164148 ; ou seja, na forma percentual → (1,3164-1)*100 = **31,64 %**

Repare que neste caso os 90% (0.90) é aplicado sobre a variação de cada dia do CDI. Portanto, o saldo fica valorizado a 31,64% do valor aplicado.

Situação 2. Acumulado

Valores de CDI e acumulados para três dias úteis :

Data	Valor do CDI	Fator de acumulação = (valor) / 100 + 1
08/08	10 %	10/100+1=1.10
09/08	12 %	12/100+1=1.12
10/08	20 %	10/100+1=1.10

Tabela XI - Valores de CDI e acumulados na situação 2

O acumulado para os três dias neste caso é : 1,100*1,120*1,100 = 1,3552 ; ou seja, na forma percentual → (1,3552-1)*100 = 35,52%, mas como queremos 90% do acumulado, temos 35,52*0,90= **31,97 %.**

Neste caso os 90% (0.90) é aplicado somente no final sobre o total acumulado. Portanto, o saldo fica valorizado a 31,97% do valor aplicado, diferente do item anterior.

1.3.1.3 O Componente de Correção Monetária

Seja uma aplicação, por exemplo, que tenha valorizado 3% em um mês (seu principal teve uma remuneração de 3%). Isto não quer dizer que o "ganho real" da aplicação seja 3%, pois é necessário descontar a inflação do período.

Assim, se a inflação neste mês foi de 2,5%, o ganho real é de apenas 0,5%.

Se a inflação tivesse sido de 3,5%, haveria perda de 0,5% neste mês.

A parcela de correção monetária é exatamente o quanto o principal teria valorizado, se tivesse rendido exatamente a inflação do período.

O índice econômico escolhido para determinar a inflação deverá ter um respaldo legal, ou seja, deverá ser aceito como índice de correção monetária para efeitos fiscais e contábeis.

No exemplo acima, a parcela de correção monetária de sua aplicação seria o seu principal vezes 0,025 (que é 2,5%). Tudo aquilo que se ganhou a mais que isso são juros reais, tudo o que se ganhou a menos que isso é perda para a inflação.

Juros Reais, portanto, representam tudo que a operação rendeu além do valor aplicado e da correção monetária incidente sobre este valor (apurada por um determinado índice econômico a escolher).

1.3.1.4 O Componente de Juros

O componente de juros é a valorização total da operação menos a correção monetária. Ele é o ganho real de uma aplicação, ou o custo real de uma captação.

Pelo sinal do componente de juros, pode-se determinar o comportamento de uma operação frente ao mercado financeiro, conforme indicado, a seguir, na tabela XII.

	Aplicação	*Captação*
Juros positivo	Ganho real	Custo real
Juros negativo	Perda para a inflação	Ganho da inflação

Tabela XII - Comportamento de um operação

Para calcular a parcela de juros, deve-se seguir o seguinte procedimento:

1. Calcular a valorização total da operação (veja acima);
2. Calcular a parcela de correção monetária (veja acima);
3. Calcular: Juros = Valorização Total - Correção Monetária.

1.3.1.5 O Componente de Encargos Financeiros

Os encargos financeiros são impostos que são cobrados cada vez que se efetua uma operação financeira. Estes impostos são diversos, e calculados das mais diversas formas. Para cada tipo de operação existe um encargo diferente. O governo federal cria estes impostos visando:

- Evitar a especulação financeira;
- Arrecadar fundos;
- Incentivar a produção;
-Incentivar aplicações de longo prazo para aumentar os investimentos.

Para satisfazer os três primeiros itens, foram criados encargos como o IR ou IRRF (Imposto de Renda Retido na Fonte), que taxam todas as aplicações que apresentam ganho real acima da correção monetária do período (um percentual sobre este ganho real). Para incentivar aplicações de longo prazo, foi criado o IOF (Imposto sobre Operações Financeiras), que taxa as aplicações de acordo com o número de dias que permanecem sem serem resgatadas (quanto mais tempo a operação permanecer sem resgates, menos imposto ela pagará). Outros instrumentos podem ser criados para incentivar aplicações de longo prazo, como a exigência de um período de carência para resgate das operações. Se a operação for resgatada dentro do período de carência, por exemplo, ela perderá todos os rendimentos.

A carência do principal é o número mínimo de dias que a aplicação deve ficar no banco para render. Um exemplo são os fundos de renda fixa, que têm uma carência de 28 dias corridos. Caso um fundo seja resgatado antes deste prazo, todo o seu rendimento será perdido.

Os encargos são divididos em 2 (dois) tipos:

Encargos Fixos: são encargos cuja alíquota (taxa) a ser aplicada para o seu cálculo é única e independente do número de dias úteis ou corridos em que a operação esteve em aberto (sem resgates).
Um exemplo de encargo fixo é o IR, pois é uma taxa fixa que independe do número de dias da operação financeira.

Encargos Variáveis: são encargos cuja alíquota (taxa) a ser aplicada para o seu cálculo, depende do número de dias úteis ou corridos em que a operação esteve em aberto (sem resgates).

Um exemplo de encargo variável é o IOF, pois é uma taxa que depende do número de dias da operação financeira.

Basicamente os encargos, fixos ou variáveis, são compostos de alíquota e base. Alíquota é a taxa percentual que vai ser aplicada sobre uma base para a determinação do valor do encargo. A base por sua vez, pode ser uma dentre os seguintes:

Base para cálculo de Encargos
Principal
Juros
Correção Monetária
Principal + Juros
Principal + Correção
Principal + Juros + Correção
Juros + Correção

Tabela XIII - Base de cálculo para Encargos

ANEXO D– CASOS DE USO

1. Especificação do Caso de Uso **"Determinar Tipos de Componentes de Cálculo"**

1.1 Identificador do Caso de Uso

001

1.2 Nome

Determinar Tipos de Componentes de Cálculo.

1.3 Descrição

Este Caso de Uso permite que o ator cadastre tipos de componentes para cálculo de Operações Financeiras.

Exemplo: Para uma Operação Financeira de Captação poderemos ter os seguintes Tipos de Componentes:

TXJUR : Valor de Taxa de Juros
SALDO : Saldo Devedor de Parcelas de Operações Financeiras
AMORT : Valor de Amortização de Principal
JUROS : Valor do Cálculo de Juros de Parcelas

1.4 Casos de Uso Relacionados

Não há Casos de Uso relacionados.

1.5 Pré-condição

1. Para este caso de uso iniciar, o ator deve estar logado no sistema e ter acesso à aplicação.

1.6 Atores

Cliente, caso o sistema seja acessado pela Internet.
Analista Financeiro, de empresas que adquirirem o produto, caso o sistema seja acessado pela Intranet.

1.7 Curso Normal dos Eventos

O ator seleciona a opção "Tipos de Componentes para Cálculo de Operações Financeiras" pelo *Menu* do Sistema.

O sistema mostra um formulário com informações necessárias para o cadastro:

- Código
- Descrição
- Característica
  - Valor Base

    Indica que os componente desse tipo devem possuir um valor informado pelo usuário em uma Simulação ou Contrato de Operação Financeira.

  - Valor Derivado de Fórmula

    Indica que os componentes desse tipo são gerados a partir de fórmulas, sendo derivados dos componentes de tipo Valor Base ou de outros componentes derivados.
- Tipo da Operação

O ator digita e salva as informações.

Caso de uso se encerra.

1.8 Pós-Condição

Tipo de Componente criado no sistema, poderão ser então criados os componentes de cálculo no sistema referentes a esse tipo de componente.

1.9 Exceções

No item 1, poderá ser apresentada a seguinte mensagem:

"Impossibilidade de alterar Valores Base Nativos do sistema"

O sistema possui Valores Base previamente cadastrados, que poderão ser utilizados em Gabaritos de Cálculo, sendo apresentados automaticamente na Simulação de uma Modalidade de Operação Financeira específica, não podendo ser , portanto, excluídos ou alterados.

1.10 Interfaces

Interface gráfica amigável entre o Caso de Uso e o Ator através da Intranet ou Internet.
Poderá ser efetuada carga dos dados do usuário que estejam em planilha Excel ou arquivo txt.

2. Especificação do Caso de Uso "**Especificar Componentes de Cálculo**"

2.1 Identificador do Caso de Uso

002

2.2 Nome

Especificar os componentes de cálculo utilizados em operações financeiras.

2.3 Descrição

Este Caso de Uso permite que o ator cadastre componentes de cálculo de Operações Financeiras.

Exemplo: Para uma Operação Financeira de Captação poderemos ter os seguintes Componentes:

TXFAT 01 : Valor de Taxa de Juros

PREST 01 : Prestações de Principal

AMORT 01 : Amortizações de Principal iguais nos períodos

JUROS 01 : Juros Mensal Linear

JUROS 02 : Juros Mensal Composto

Note que poderão existir vários componentes de um mesmo tipo.

2.4. Casos de Uso Relacionados

003 – Consultar Palavras Reservadas utilizadas em Cálculo

004 – Cadastrar Condições de Cálculo

2.5. Atores

Cliente, caso o sistema seja acessado pela Internet.

Analista Financeiro, de empresas que adquirirem o produto, caso o sistema seja acessado pela Intranet.

2.6. Pré-condição

1. Para este caso de uso inicar, o ator deve estar logado e ter feito alguma reserva no sistema.

2. Para cadastrar um componente, o tipo do componente deve estar cadastrado.

2.7. Curso Normal dos Eventos

1. O ator seleciona "Componentes de Cálculo" pelo *Menu* do Sistema.
2. O sistema solicita as seguintes informações para cadastrar um novo Componente de Cálculo:

   - Código
   - Descrição
   - Fórmula

     É a expressão matemática para calcular o Componente. Na Fórmula podem ser referenciados:

     - Valores Base

       Exemplo:

       LIBE / 100

     - Componentes de Cálculo

       Exemplo:

       TXFAT01 * SALDO01

     - Palavras Reservadas

       Exemplo:

       SOMA_ATE_COMPONENTE(AMORT01 , PREST01 , PREST01)

     - Seqüência de Condições

       Exemplo:

SQCD 001

- Seqüência de condições

  Deve ser utilizado somente para cálculo de componentes que apresentem uma ou mais possibilidades de cálculo, seguindo as respectivas condições que devem ser previamente cadastradas.

  Para indicar o uso de uma seqüência de condições, primeiro cadastre as condições e depois relacione-as em uma ordem de verificação, para isso deverão ser digitadas as seguintes informações :

  - Ordem de Verificação
  - Código da Condição
  - Descrição

3. O ator digita e salva as informações solicitadas e o caso de uso se encerra.

2.8. Pós-condição

Componente criado no sistema, poderão ser então utilizados em Simulação de cálculo de Operações Financeiras no sistema.

2.9. Exceções

2.9.1. No item 2, poderá ocorrer a seguinte mensagem:

"Expressão inválida na Fórmula"

Caso o ator tenha utilizado na fórmula uma palavra que não seja Valor Base, Componente , Palavra Reservada ou uma Seqüência de Condições, o sistema retornará uma mensagem de erro, "Expressão Inválida na Fórmula".

2.9.2. No item 3, poderá ocorrer a seguinte mensagem:

"Visão Geral da Sequencia de Condições"

Haverá um atalho (botão, por exemplo) para o ator consultar a composição geral da seqüência de condições.

Exemplo:

SE (..... SE (..... OU ....) E .....

2.10. Interfaces

Interface gráfica amigável entre o Caso de Uso e o Ator através da *Intranet* ou *Internet*.

3. Especificação do Caso de Uso "Verificar Palavras Reservadas"

3.1. Identificador de Caso de Uso

003

3.2. Nome

Consultar Palavras Reservadas do sistema que podem ser utilizadas em fórmula de cálculo dos componentes de parcelas de Operações Financeiras.

3.3. Descrição

Este Caso de Uso permite que o ator consulte as Palavras Reservadas existentes no sistema, e que podem ser utilizadas em fórmulas de componentes de cálculo de Operações Financeiras. O sistema apresenta uma explicação sobre a Palavra Reservada e como deve ser utilizada, ou seja, caso possua argumentos que devam ser utilizados como parâmetros, serão especificados um a um.

Conceito : Para que sejam acumulados os juros mês a mês de várias parcelas, utilizar a seguinte palavra reservada:

SOMA_ATE_COMPONENTE (ARG1 , ARG2 , ARG3) onde:
Arg1 : Componente a ser acumulado
Arg2 : Componente a partir de onde a somatória começa
Arg3 : Componente que zera a somatória

Exemplo de Utilização :

A fórmula do componente JUROS ACM (Juros Acumulados) é :

SOMA_ATE_COMPONENTE (JUROS SMP, PREST ENC , PREST ENC)

Explicação : desta forma o componente JUROS ACM é acumulado mês a mês de uma prestação de Encargos (PREST ENC) a outra.

3.4. Casos de Uso Relacionados

Usado pelo Caso de Uso 002.

3.5. Atores

Cliente, no caso do sistema ser acessado pela *Internet*.
Analista Financeiro, de empresas que adquirirem o produto, no caso do sistema ser acessado pela *Intranet*.

3.6. Pré-condição

Para este caso de uso inicar, o ator deve estar logado e possuir acesso a aplicação.

3.7. Curso Normal dos Eventos

a. O ator seleciona a opção "Consultar Funções do Sistema" pelo *Menu* do Sistema.

b. O sistema mostra um formulário com as informações das funções do sistema:

- Código da Função
- Descrição da Função
- Ajuda(Help) sobre a utilização da Função de Sistema em fórmulas de componentes.

c. O ator visualiza as informações e o caso de uso se encerra.

3.8. Pós-condição

Não há.

3.9. Exceções

Não há.

3.10. Interfaces

Interface gráfica amigável entre o Caso de Uso e o Ator através da *Intranet* ou *Internet*.

4. Especificação do Caso de Uso "Determinar Condições de Cálculo"

4.1 Identificador do Caso de Uso

004

4.2 Nome

Cadastrar Condições de Cálculo de componentes de parcelas de Operações Financeiras.

4.3 Descrição

Este Caso de Uso permite que o ator especifique cálculos financeiros que sejam condicionais e possuam várias alternativas de cálculo.

4.4 Casos de Uso Relacionados

Caso de Uso 003

4.4 Atores

Cliente, no caso do sistema ser acessado pela *Internet*.

Analista Financeiro, de empresas que adquirirem o produto, no caso do sistema ser acessado pela *Intranet*.

4.5 Pré-Condição

Para este caso de uso iniciar, o ator deve estar logado e ter acesso a aplicação.

4.6 Curso Normal dos Eventos

O ator seleciona "Cadastrar Condições de Cálculo" pelo *menu* do sistema.
Sistema solicita as seguintes informações:

- Código da Condição
- Descrição da Condição
- Tipo do primeiro atributo a ser testado

O tipo do atributo pode ser :

- Valores Base
- Componente de Cálculo
- Palavra Reservada

- Primeiro atributo a ser testado
- Operador Condicional
- Tipo do segundo atributo a ser testado

  O tipo do atributo pode ser :

  - Valores Base
  - Componente de Cálculo
  - Palavra Reservada
  - Valor
- Segundo atributo a ser testado
- Operador Lógico

  O operador lógico pode ser :

  - É IGUAL A
  - É MAIOR QUE
  - É MAIOR OU IGUAL A
  - É MENOR QUE
  - É MENOR OU IGUAL A
  - COMEÇA COM
  - É DIFERENTE DE
  - FIM

  O operador lógico "FIM" indica o término da condição quando for composta.

Usuário cria e salva a condição.

O sistema retorna com a confirmação e o caso de uso se encerra.

4.7 Pós-Condição

Condições de cálculo cadastradas e aptas para serem utilizadas pelas seqüências de condições.

4.8 Exceções

Não há.

4.9 Interfaces

Interface gráfica amigável entre o Caso de Uso e o Ator através da *Intranet* ou *Internet*.

5. Especificação do Caso de Uso "Simular Captações"

5.1 Identificador do Caso de Uso

004

5.2 Nome

Simular Operações Financeiras de Captação.

5.3 Descrição

Este Caso de Uso permite que o ator consiga simular uma Operação Financeira de Captação.

5.4 Casos de Uso Relacionados

Caso de Uso 003

5.5 Atores

Cliente, no caso do sistema ser acessado pela *Internet*.

Analista Financeiro, de empresas que adquirirem o produto, no caso do sistema ser acessado pela *Intranet*.

5.6 Pré-Condição

Para este caso de uso iniciar, o ator deve estar logado e ter acesso a aplicação. Devem estar previamente cadastrados : Tipos de Componentes de Cálculo, Componentes de Cálculo, Condições de Cálculo.

5.7 Curso Normal dos Eventos

1. O ator seleciona "Simular Operações de Captação" pelo *menu* do sistema.
2. Sistema solicita as seguintes informações:

- Vigência da Operação
    - Valor da Operação

- Data Base das parcelas
- Freqüência de Cálculo
- Moeda
- Modalidade da Captação
- Valores Base

  Apresentados pelo *Wizard* :

  - Valor de Taxa de Juros (Obrigatório)
  - Valor de Taxa de Comissão
  - Valor de Taxa de Impostos
  - Valor de Taxa de Administração

  Usuário poderá acrescentar mais valores base. Esses valores base podem ser nativos do sistema ou terem sido criados pelo usuário.

- Gabarito de Modelo de Cálculo (Opcional)

3. Sistema apresentará uma espécie de planilha constituindo-se de uma matriz com o cruzamento das informações PERÍODOS x COMPONENTES:

   - Períodos

     Cada linha da matriz constitui uma data de cálculo, onde um evento (componente) de cálculo deva ocorrer.

   - Componentes de Cálculo

     Cada coluna da matriz constitui um componente de cálculo, semelhante aos cálculos financeiros feitos em planilha Excel, onde em uma coluna , por exemplo, temos os valores de amortização de Principal , outra coluna apresenta os valores de juros , outra de encargos, impostos, etc , e por fim uma coluna com saldo devedor.

     Procedimentos de trabalho com a "Planilha"

     No sistema teremos uma "planilha" de cálculo .

Verificar o anexo J de IHM.

Perceba que do lado da coluna de períodos de cálculo, temos alguns índices nomeados pelas colunas de nome: Instante, Ocorrência, Parcela e Data. São índices que poderão ser utilizados como auxiliares de fórmulas instantâneas criadas no simulador, ou seja, são como se fossem componentes "temporários" , feitos sob medida para a planilha que está sendo trabalhada e elaborada no momento.

Procedimento para utilizar esses índices:

Do lado direito do componente, coloque imediatamente um parênteses com a "localização" do componente pelo indicador/índice/localizador desejado, utilizando as seguintes letras para representá-los:

C : para localizar por ordem de Componente.
D : para localizar por ordem de Data.
P : para localizar por ordem de Parcela.
I : para localizar por ordem de Instante.
Exemplo:

Se em uma "célula" resolvermos efetuar um cálculo utilizando as células azul e verde utilizando uma fórmula qualquer, teríamos que fazer referência a seguinte nomenclatura :

TXFAT001 (D10) * SALDO002 (D7)

Pela fórmula acima o sistema entenderá que é para multiplicar o componente "TXFAT001" da décima data , portanto ocorrendo em 27/12/2002 pelo componente "SALDO002" da sétima data, portanto ocorrendo em 15/11/2002.

Observação Importante

Caso já tenham sido feitas fórmulas instantâneas utilizando-se como indexadores as datas e/ou parcelas de cálculo criadas até determinado momento, e sejam posteriormente incluídas novas datas pelo usuário e/ou geradas novas parcelas pelo sistema, as referências são automaticamente alteradas , quando necessário.

Exemplo:

A célula que tinha a seguinte fórmula:

TXFAT001 (D10) * SALDO002 (D7)

Caso seja incluída por exemplo, a data 01/11/2002 que será a sétima data da planilha, perceba que a data 15/11/2002 referenciada por "D7" passará a ser referenciada por "D8" pois passou a ser a oitava data do sistema. A data 27/12/2002 também não é mais a décima data do sistema, e sim décima primeira data , e não será referenciada em fórmulas por "D10" e sim por "D11". Portanto, a célula amarela passa a ter a seguinte fórmula instantânea automaticamente, alterada pelo sistema:

TXFAT001 (D11) * SALDO002 (D8)

4. Após o usuário ter finalizado o trabalho de configuração da planilha, deverá solicitar a opção de "SIMULAR VALORES", ou simplesmente salvar a configuração da Simulação, para a qual o sistema dará um número de identificação para que possa ser resgatada novamente.

5. O sistema apresenta a planilha de Planejamento do Cálculo completo da Operação Financeira de Captação contendo somente valores; no entanto, o sistema armazena a Planilha de Configuração, que é a Planilha que

possui somente os cálculos, podendo ser alterada em futuras simulações pelo usuário.

6. Caso de Uso se encerra.

5.8 Pós-Condição

Condições de cálculo cadastradas e aptas para serem utilizadas pelas seqüências de condições.

5.9 Exceções

Não há.

5.10 Interfaces

Interface gráfica amigável entre o Caso de Uso e o Ator através da *Intranet* ou *Internet.*

ANEXO E – DIAGRAMA DE CLASSE

Abaixo, está apresentado o diagrama de classe do estudo de caso.

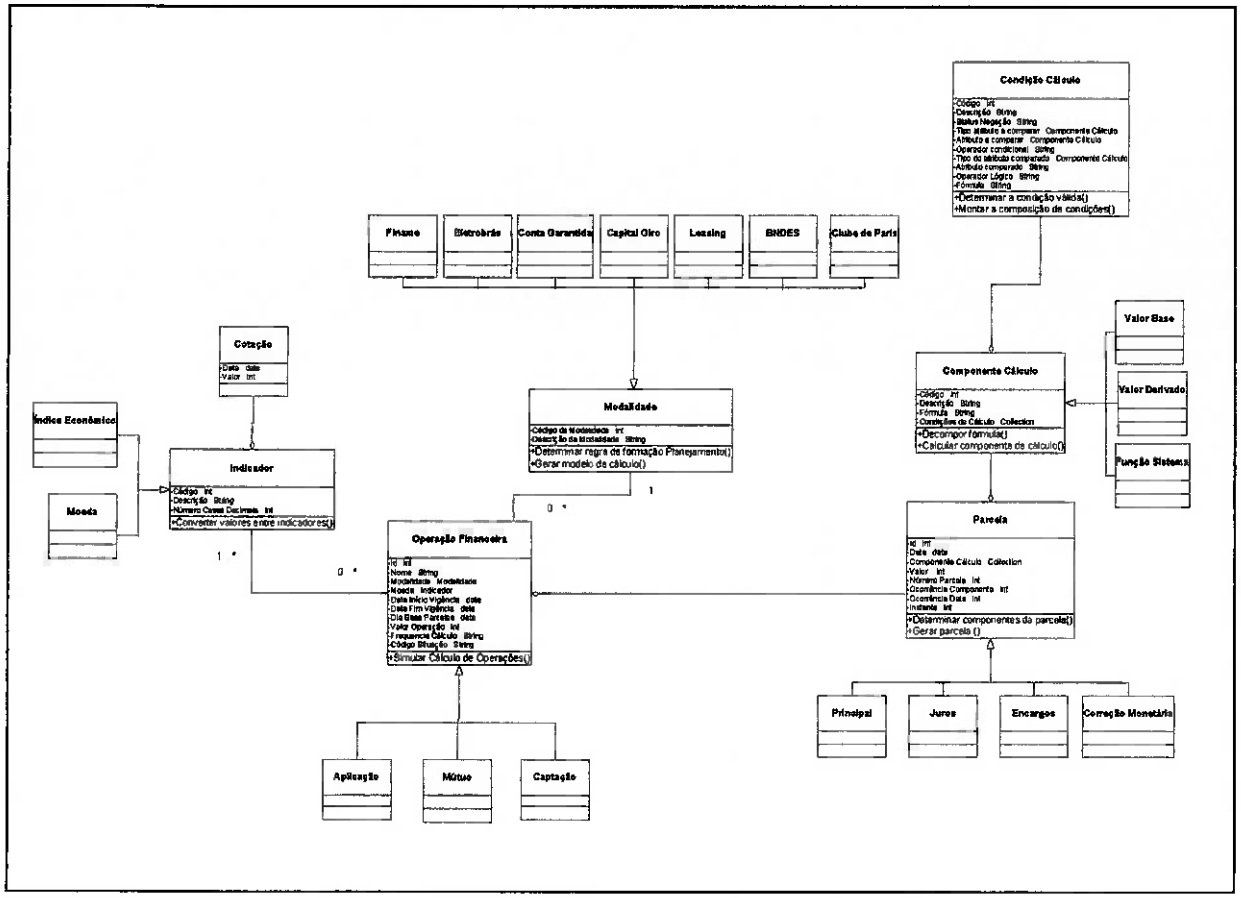
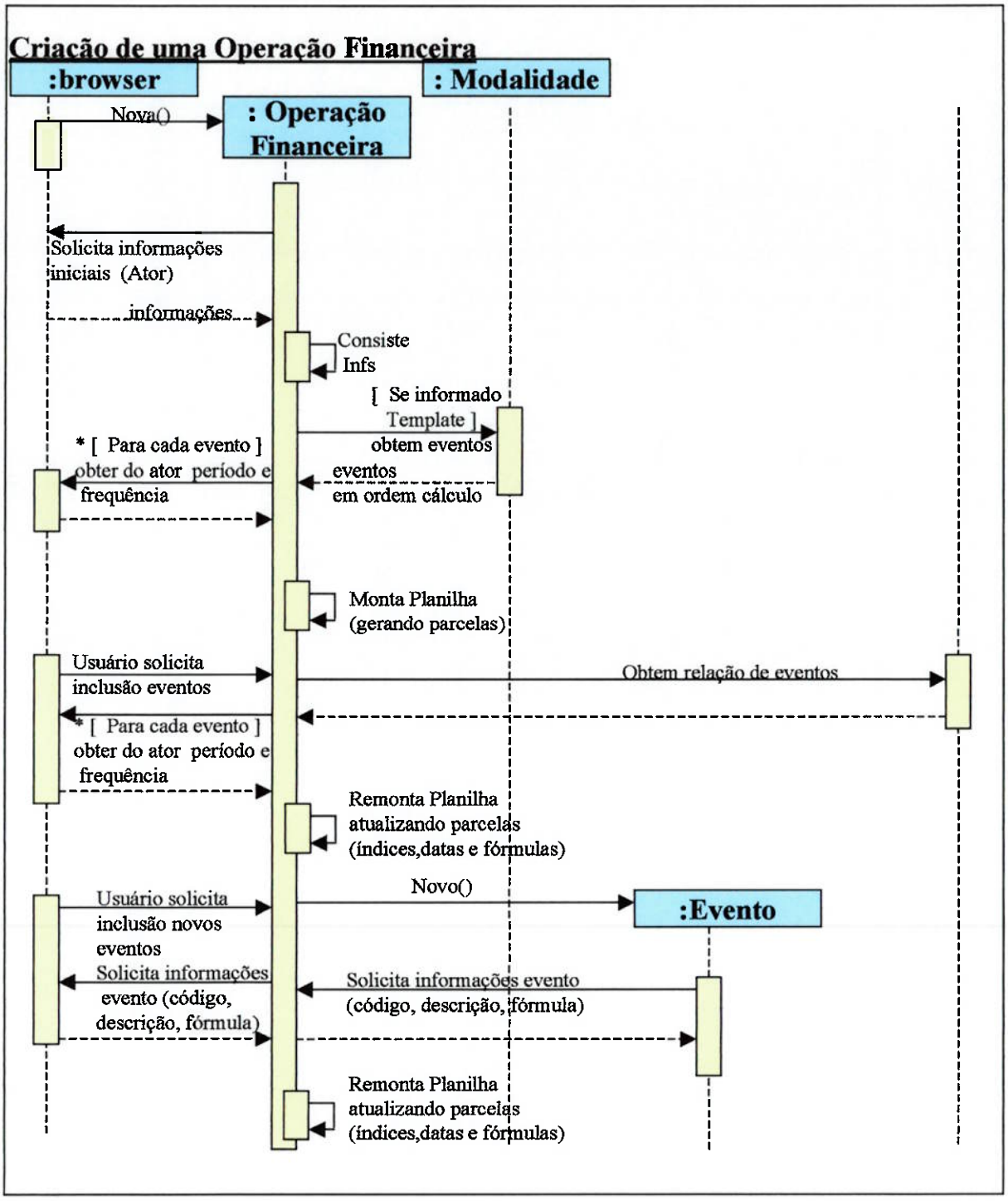


Figura 30 - Exemplo de Diagrama de Classe do Estudo de Caso

ANEXO F – DIAGRAMA DE SEQÜÊNCIA



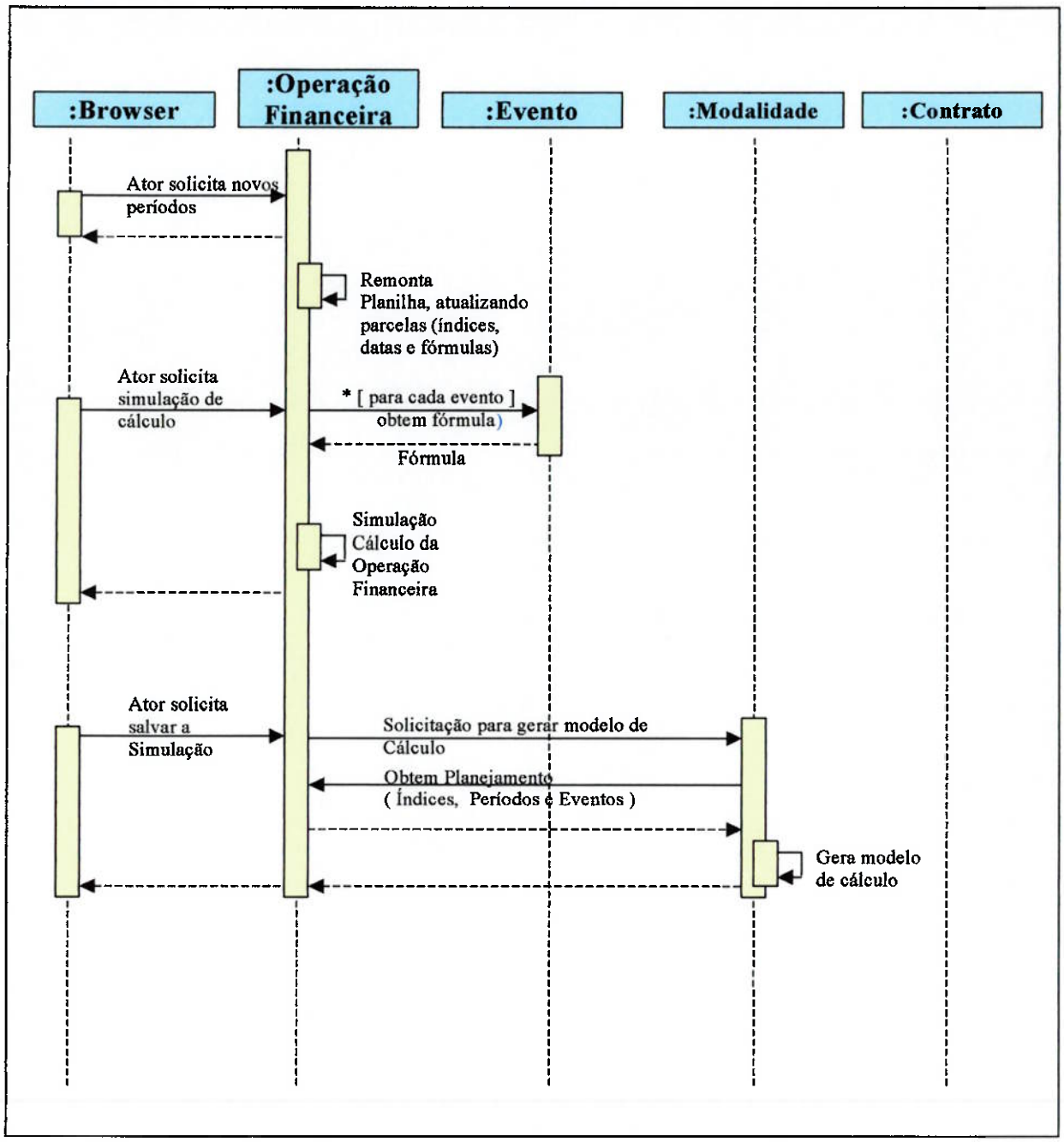


Figura 31 - Diagrama de Seqüência do Estudo de Caso

ANEXO G – DIAGRAMA DE ESTADOS

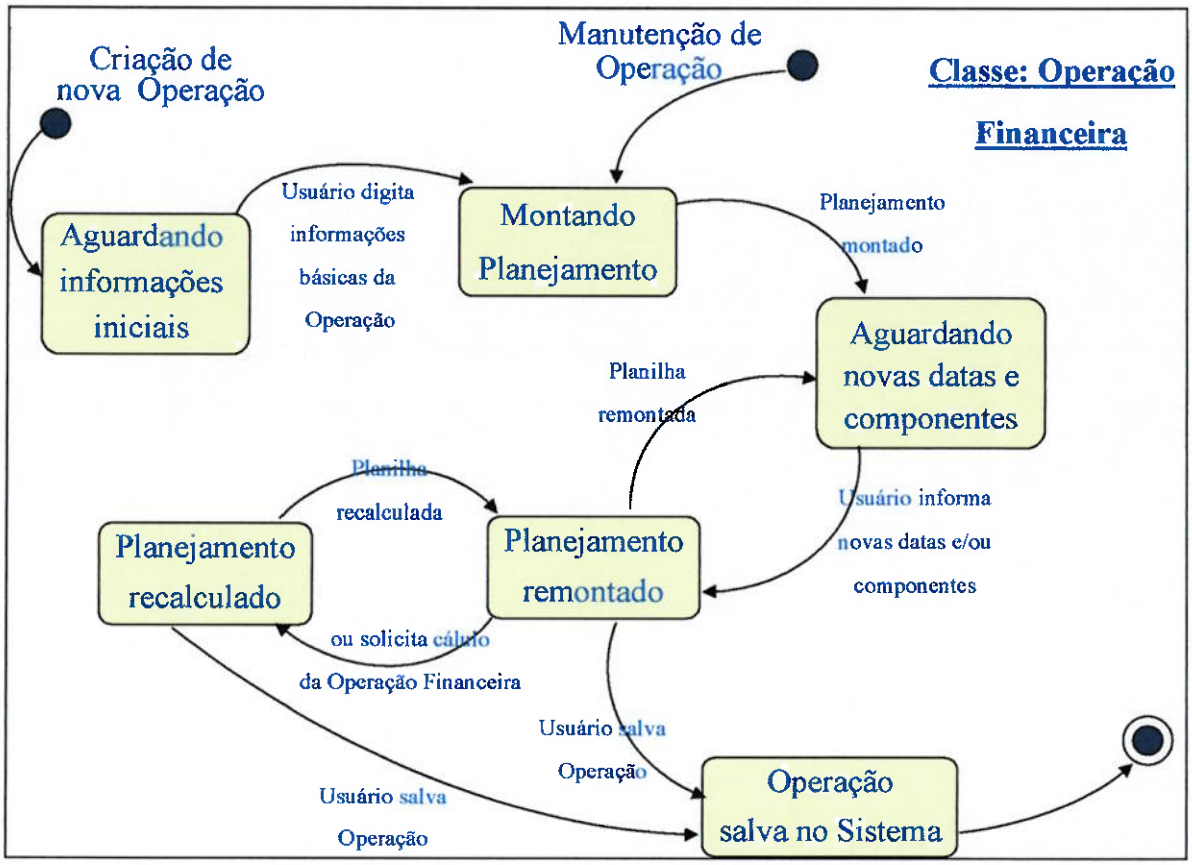


Figura 32 - Diagrama de Estados do Estudo de Caso

Mapeamento dos Estados para os atributos das classes:

Classe Operação Financeira

Estado	Representação do Estado
Aguardando informações iniciais	Código Situação = IN
Montando Planejamento	Código Situação = MN
Aguardando novas datas e componentes de cálculo	Código Situação = MT
Planejamento remontado	Código Situação = AL
Planejamento recalculado	Código Situação = CL
Operação salva no sistema	Operação gravada na tabela OPER_FINANCEIRA

ANEXO H – DIAGRAMA ENTIDADE-RELACIONAMENTO

A seguir o Diagrama Entidade-Relacionamento do estudo de caso e o dicionário de dados com a descrição das tabelas e atributos.

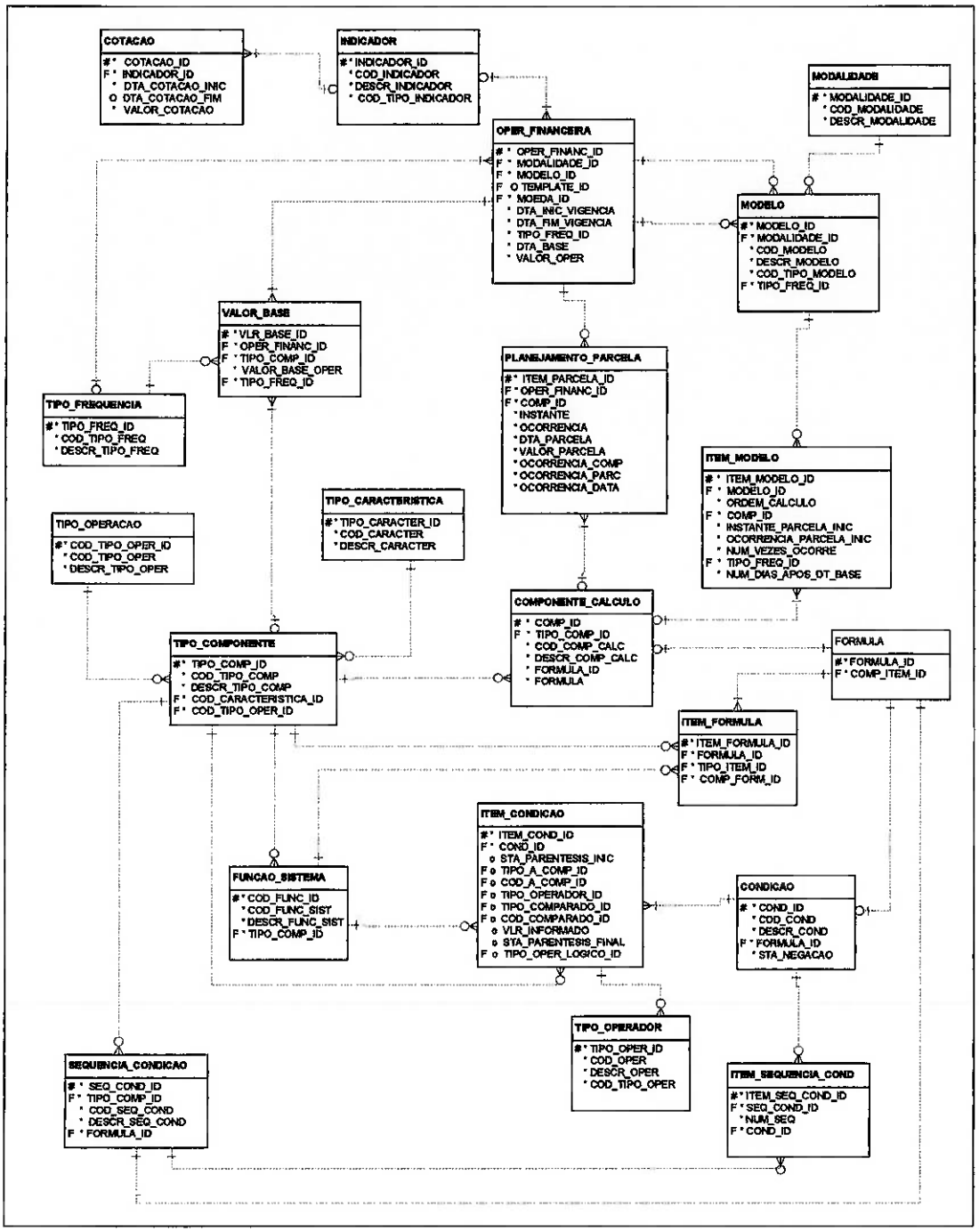


Figura 33 - DER do Estudo de Caso

Dicionário de dados sobre as entidades e atributos, em ordem alfabética:

COMPONENTE_CALCULO

Representa cada componente de cálculo das parcelas do planejamento de uma operação financeira.

COMP_ID	Identificação do componente de cálculo. Chave Primária
TIPO_COMP_ID	Identificação do tipo de componente de cálculo. Chave Estrangeira para a tabela TIPOS_COMPONENTES.
COD_COMP_CALC	Código do componente de cálculo atribuído pelo usuário. Exemplo: JUR01
DESCR_COMP_CALC	Descrição do componente de cálculo. Exemplo: JUROS SIMPLES
FORMULA_ID	Identificação da fórmula. Chave Estrangeira para a tabela FORMULA.
FORMULA	Data de início da vigência da operação financeira.

CONDICAO

Representa as condições de cálculo de um determinado componente de cálculo de uma parcela da operação financeira.

COND_ID	Identificação da condição. Chave primária.
COD_COND	Código de identificação da condição de cálculo definido pelo usuário.
DESCR_COND	Descrição da condição.
FORMULA_ID	Identificação da fórmula da condição. Chave estrangeira para a tabela FORMULA.
STA_NEGACAO	Status de identificação para que seja testada a condição negação da fórmula. É para ser verificado se NOT(condição) é verdadeira.

COTACAO

Representa as cotações de um indicador em determinado período.

COTACAO_ID	Identificador da cotação. Chave primária.
INDICADOR_ID	Identificador do indicador. Chave estrangeira para a tabela INDICADOR.
DTA_COTACAO_INIC	Data inicial do período da cotação.
DTA_COTACAO_FIM	Data final do período da cotação.
VALOR_COTACAO	Valor da cotação do indicador.

FORMULA

Representa uma tabela de relacionamento entre os itens de fórmula e as entidades que possuem fórmula cadastrada, que podem ser componente de cálculo, condição ou seqüência de condições.

FORMULA_ID	Identificação da fórmula. Chave primária.
COMP_ITEM_ID	Identificação da entidade que possui a fórmula. Pode ser um componente de cálculo, uma condição ou uma seqüência de condições. Desta forma, o relacionamento ocorre da seguinte forma: Campo COMP_ITEM_ID pode corresponder ao campo COMP_ID da tabela COMPONENTES_CALCULO, ou ao campo COND_ID da tabela CONDICOES, ou ao campo SEQ_COND_ID da tabela SEQUENCIA_CONDICOES.

FUNCAO_SISTEMA

Representa cada função de sistema com cálculos pré-determinados.

COD_FUNC_ID	Identificador da função do sistema. Chave primária
COD_FUNC_SIST	Código de identificação da função do sistema a ser utilizado pelo usuário em fórmulas de componentes de cálculo de parcelas de operações financeiras. Exemplo: OCORRIDO
DESCR_FUNC_SIST	Descrição da função.
TIPO_COMP_ID	Identificação de tipo de componente de cálculo. Chave Estrangeira para a tabela TIPOS_COMPONENTES. Quando esse código de função de sistema constar em uma fórmula de cálculo, através desse campo TIPO_COMP_ID, será possível identificar que se trata de uma função de sistema.

INDICADOR

Representa cada item de condição de cálculo de um componente

INDICADOR_ID	Identificação de um indicador. Chave primária.
COD_INDICADOR	Código do indicador cadatrado pelo usuário.
COD_TIPO_INDICADOR	Tipo do indicador. Pode ser 'MD' para moeda ou 'IN' para índice.
DESCR_INDICADOR	Descrição do indicador.

ITEM_CONDICAO

Representa cada item de condição de cálculo de um componente

ITEM_COND_ID	Identificação do item da condição de cálculo. Chave primária.
COND_ID	Identificação da condição de cálculo. Chave estrangeira para a tabela CONDICOES.
STA_PARENTESIS_INIC	Identificação se deverá ser aberto um parêntesis. Essa identificação deve ser usada para condições compostas complexas.
TIPO_A_COMP_ID	Identificação do tipo do componente a ser comparado, ou seja, o primeiro atributo. Chave estrangeira para a tabela TIPOS_COMPONENTES.
COD_A_COMP_ID	Identificação do componente a ser comparado. Pode ser um componente de cálculo, uma função de sistema ou uma outra condição. Desta forma, o relacionamento ocorre da seguinte forma: Campo COD_A_COMP_ID pode corresponder ao campo COMP_ID da tabela COMPONENTES_CALCULO, ou ao campo COND_ID da tabela CONDICOES, ou ao campo COD_FUNC_ID da tabela FUNCOES_SISTEMA.
TIPO_OPERADOR_ID	Identificação do tipo do operador condicional. Chave estrangeira para a tabela TIPO_OPERADOR.
TIPO_COMPARADO_ID	Identificação do tipo do componente com o qual se quer comparar, ou seja, o segundo atributo. Pode ser uma função de sistema, um componente de cálculo que seja valor base ou valor derivado, ou ainda uma constante(valor informado). Caso esteja sendo testada uma condição, esse campo não deve ser informado, pois o tipo de operador será verdadeiro/falso, portanto, não existe o segundo atributo.
COD_COMPARADO_ID	Identificação do componente a ser comparado. Pode ser um componente de cálculo, uma função de sistema ou uma outra condição. Desta forma, o relacionamento ocorre da seguinte forma: Campo COD_A_COMP_ID pode corresponder ao campo COMP_ID da tabela COMPONENTES_CALCULO, ou ao campo COND_ID da tabela CONDICOES, ou ao campo COD_FUNC_ID da tabela FUNCOES_SISTEMA. Caso esteja sendo testada uma condição, esse campo não deve ser informado, pois o tipo de operador será verdadeiro/falso, portanto, não existe o segundo atributo.

VLR_INFORMADO	Valor com o qual se quer comparar o primeiro atributo da cláusula de comparação, este campo estará preenchido somente quando a comparação for feita com constantes. Esse campo não poderá estar preenchido se o campo COD_COMPARADO_ID estiver preenchido, e vice-versa.
STA_PARENTESIS_FINAL	Identificação se deverá ser fechado um parêntesis. Essa identificação deve ser usada para condições compostas complexas.
TIPO_OPER_LOGICO_ID	Identificação do tipo de operador lógico. Chave estrangeira para a tabela TIPO_OPERADOR.

ITEM_FORMULA

Representa cada componente que compõe a estrutura de uma fórmula.

ITEM_FORMULA_ID	Identificação do item da fórmula. Chave primária.
FORMULA_ID	Identificação da fórmula. Chave estrangeira para a tabela FORMULA.
TIPO_ITEM_ID	Identificação do tipo do item da fórmula. Chave estrangeira para a tabela TIPOS_COMPONENTES. Esse campo serve para identificar se o componente da fórmula é outro componente, ou uma seqüência de condições ou uma função de sistema.
COMP_FORM_ID	Identificação do item da fórmula. Esse campo possui o código do componente. Desta forma, o relacionamento ocorre da seguinte forma: Campo COMP_FORM_ID pode corresponder ao campo COMP_ID da tabela COMPONENTES_CALCULO, ou ao campo COD_FUNC_ID da tabela FUNCOES_SISTEMA, ou ao campo SEQ_COND_ID da tabela SEQUENCIA_CONDICOES. É importante destacar que uma condição não poderá fazer parte de uma fórmula, somente a seqüência de condições.

ITEM_MODELO

Representa cada item de um modelo de captação.

ITEM_MODELO_ID	Identificação do item de modelo. Chave primária.
MODELO_ID	Identificação do modelo de cálculo da operação financeira de captação. Chave estrangeira para a tabela MODELO.
ORDEM_CALCULO	Ordem de cálculo do componente de cálculo no modelo.
COMP_ID	Identificação do componente. Chave estrangeira para a tabela COMPONENTE_CALCULO.
INSTANTE_PARCELA_INIC	Instante de parcela inicial de cálculo.
OCORRENCIA_PARCELA_INIC	Ocorrência da parcela inicial de cálculo.
NUM_VEZES_OCORRE	Número de vezes que o componente ocorre a partir da parcela inicial e freqüência especificados no item.
TIPO_FREQ_ID	Identificação da freqüência em que o componente ocorre. Chave estrangeira para a tabela TIPO_FREQUENCIA.
NUM_DIAS_APOS_DT_BASE	Número de dias a deslocar da data base para cálculo da data do componente na operação financeira.

ITEM_SEQUENCIA_COND

Representa o relacionamento de todas as condições que fazem parte de uma seqüência de condições de cálculo de um componente de parcela de uma operação financeira.

ITEM_SEQ_COND_ID	Identificação do item da seqüência de condições. Chave primária.
SEQ_COND_ID	Identificação da seqüência de condições. Chave estrangeira para a tabela SEQUENCIA_CONDICOES.
NUM_SEQ	Número seqüencial que indica a ordem em que as condições devem ser verificadas.
COND_ID	Identificação da condição a ser checada. Chave estrangeira para a tabela CONDICOES.

MODALIDADE

Representa os tipos de modalidades de captação.

MODALIDADE_ID	Identificador da modalidade. Chave primária.
COD_MODALIDADE	Código da modalidade. Exemplo: FINAME, ELETROBRAS, BNDES.
DESCR_MODALIDADE	Descrição detalhada da modalidade.

MODELO

Representa cada operação financeira.

MODELO_ID	Identificação do modelo. Chave primária
MODALIDADE_ID	Identificação da modalidade da captação. Chave estrangeira para a tabela MODALIDADE.
COD_MODELO	Código do modelo.
DESCR_MODELO	Descrição do modelo.
COD_TIPO_MODELO	Código do tipo de modelo. "TP"- Template ou "CT"- Contrato
TIPO_FREQ_ID	Identificação do tipo de freqüência. Chave estrangeira para a tabela TIPO_FREQUENCIA.

OPER_FINANCEIRA

Representa cada operação financeira.

OPER_FINANC_ID	Identificação da operação financeira. Chave primária.
MODALIDADE_ID	Identificação da modalidade de operação financeira de captação. Chave estrangeira para a tabela MODALIDADE.
MODELO_ID	Identificação do modelo. Chave estrangeira para a tabela MODELO.
TEMPLATE_ID	Identificação do template utilizado para determinar as parcelas da operação financeira. Chave estrangeira para a tabela MODELO.
MOEDA_ID	Identificação da moeda. Chave estrangeira para a tabela INDICADOR.
DTA_INIC_VIGENCIA	Data de início da vigência da operação financeira.
DTA_FIM_VIGENCIA	Data final da vigência da operação financeira.
TIPO_FREQ_ID	Tipo de freqüência de cálculo da operação financeira. Chave estrangeira para a tabela TIPO_FREQUENCIA.
DTA_BASE	Data base das parcelas.
VALOR_OPER	Valor da operação financeira.

PLANEJAMENTO_PARCELA

Representa cada operação financeira.

ITEM_PARCELA_ID	Identificação do item de parcela. Chave primária.
OPER_FINANC_ID	Identificação da operação financeira a qual se refere o planejamento. Chave estrangeira para a tabela OPER_FINANCEIRA.
COMP_ID	Identificação do componente da parcela. Chave estrangeira para a tabela COMPONENTE_CALCULO.
INSTANTE	Número do instante da parcela. Cada mês é um instante.
OCORRENCIA	Número seqüencial da parcela dentro do instante. Cada parcela, em ordem de data tem uma ocorrência.
DTA_PARCELA	Data do componente de cálculo na parcela.
VALOR_PARCELA	Valor do componente na parcela.
OCOR_COMP	Ordem de ocorrência do componente no planejamento.
OCOR_PARC	Ordem de ocorrência da parcela no planejamento.
OCOR_DATA	Ordem de ocorrência da data da parcela no planejamento.

SEQUENCIA_CONDICAO

Representa cada operação financeira.

SEQ_COND_ID	Identificação da seqüência de condições. Chave primária.
TIPO_COMP_ID	Identificação de tipo de componente de cálculo. Chave Estrangeira para a tabela TIPOS_COMPONENTES. Quando esse código de seqüência de condições constar em uma fórmula de cálculo, através desse campo TIPO_COMP_ID, será possível identificar que se trata de uma seqüência de condições.
COD_SEQ_COND	Código da seqüência de condições informado pelo usuário.
DESCR_SEQ_COND	Descrição da seqüência de condições.
FORMULA_ID	Identificação da fórmula que deverá ser assumida quando o resultado da verificação da seqüência de condições for falsa. Chave estrangeira para a tabela FORMULA.

TIPO_CARACTERISTICA

Representa os tipos de características de componentes de cálculo

TIPO_CARACTER_ID	Identificação dos tipos de características de componentes de cálculo. Chave primária.

COD_CARACTER	Código do tipo de característica. Os códigos possíveis são: VB – Valor Base , VD – Valor Derivado , FS – Função de Sistema, SQ – Seqüência de Condições e CD - Condição.
DESCR_CARACTER	Descrição do tipo de característica.

TIPO_COMPONENTE

Representa os tipos de componentes que podem ser utilizados em fórmulas de componentes de parcelas das operações financeiras.

TIPO_COMP_ID	Identificação do tipo de componente de cálculo. Chave primária.
COD_TIPO_COMP	Código do tipo de componente de cálculo dado pelo usuário.
DESCR_TIPO_COMP	Descrição do tipo de componente de cálculo dado pelo usuário.
COD_CARACTERISTICA_ID	Identificação da característica do tipo de componente. Chave estrangeira para a tabela TIPOS_CARACTERISTICAS.
COD_TIPO_OPER_ID	Identificação do tipo de operação que o componente faz parte. Por exemplo: Juros, Amortização, Encargos, Correção Monetária ou Saldo Devedor. Chave estrangeira para a tabela TIPOS_OPERACAO.

TIPO_FREQUENCIA

Representa os tipos de freqüência para ocorrência de componentes de cálculo.

TIPO_FREQ_ID	Identificação do tipo de freqüência.Chave primária.
COD_TIPO_FREQ	Código do tipo de freqüência.
DESCR_TIPO_FREQ	Descrição do tipo de freqüência.

TIPO_OPERADOR

Representa operadores de comparação.

TIPO_OPER_ID	Identificação do tipo de operador.Chave primária.
COD_OPER	Código do tipo de operador.
DESCR_OPER	Descrição do tipo de operador.
COD_TIPO_OPER	Tipo do operador. "LG"= Lógico ou "RL"=Relacional.

TIPO_OPERACAO

Representa tipo de operação de componentes nas parcelas de operações financeiras.

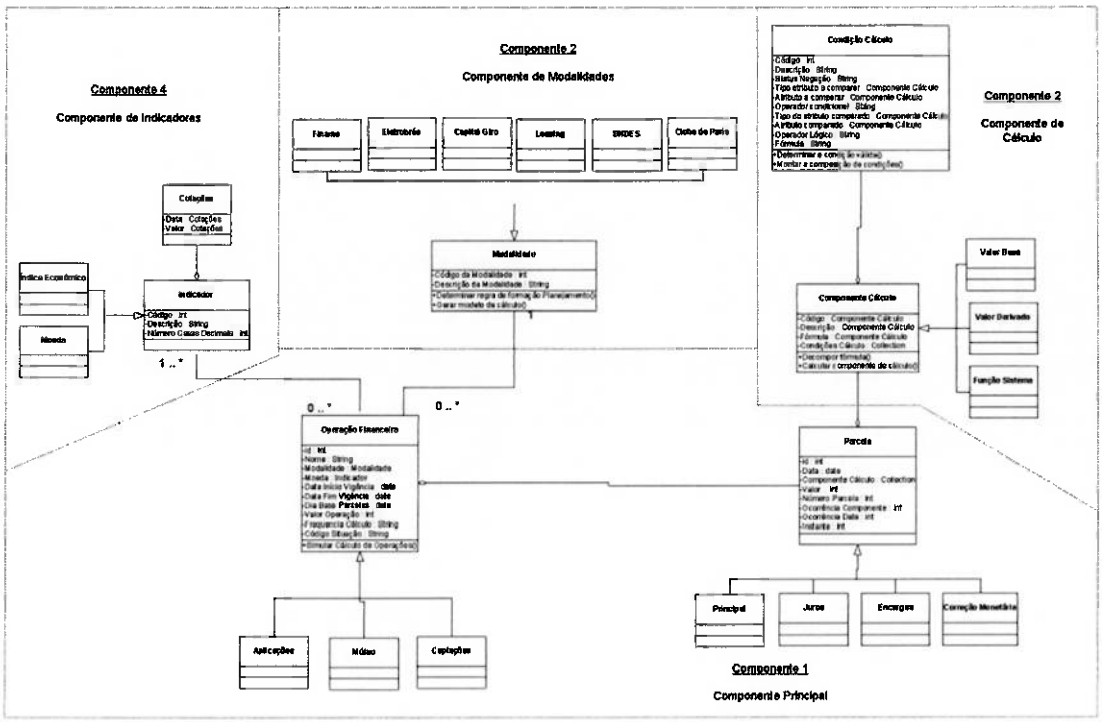
COD_TIPO_OPER_ID	Identificação do tipo de operação. Chave primária.
COD_TIPO_OPER	Código do tipo de operação. Tipos possíveis: AM – Amortização, JR – Juros, EN – Encargos, SD – Saldo Devedor, CM – Correção Monetária.
DESCR_TIPO_OPER	Descrição do tipo de operação.

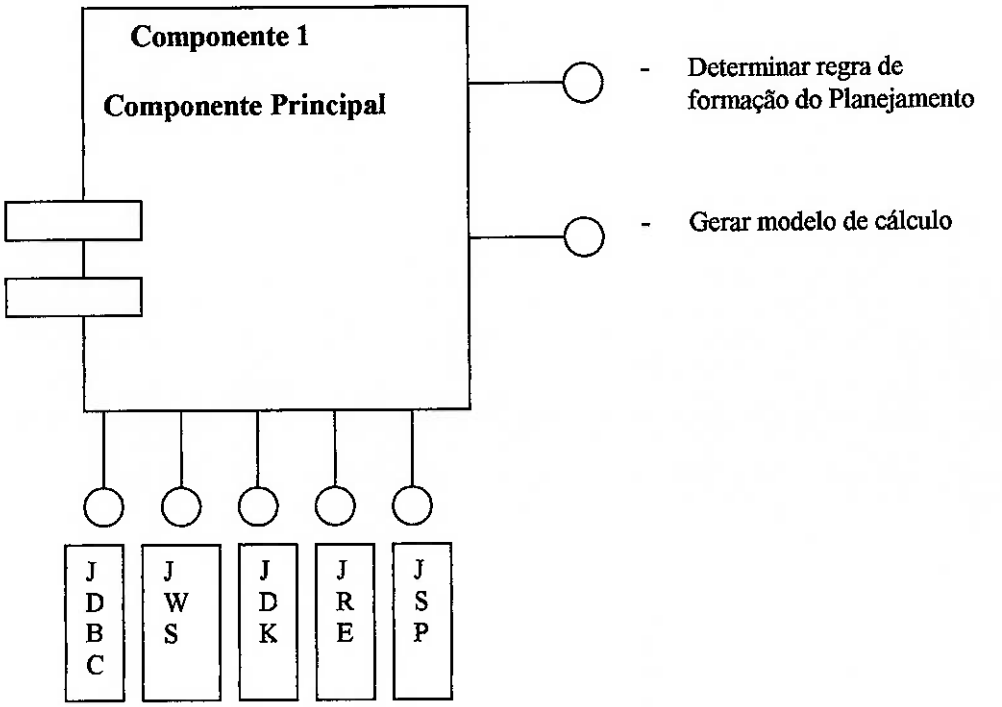
VALOR_BASE

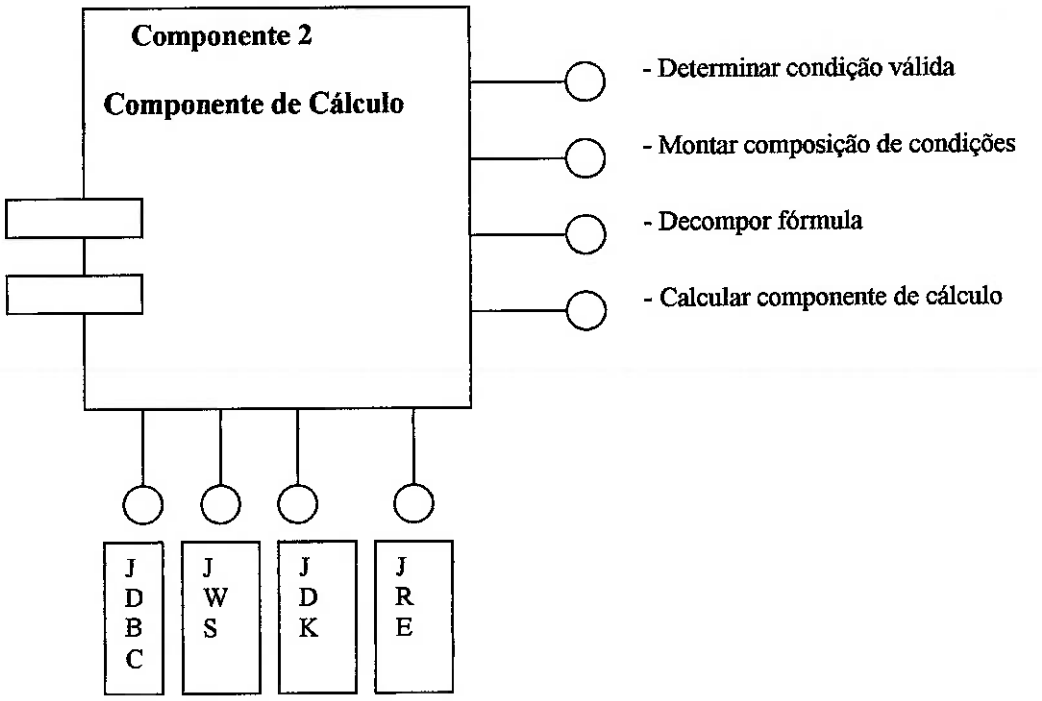
Representa os valores base de uma operação financeira.

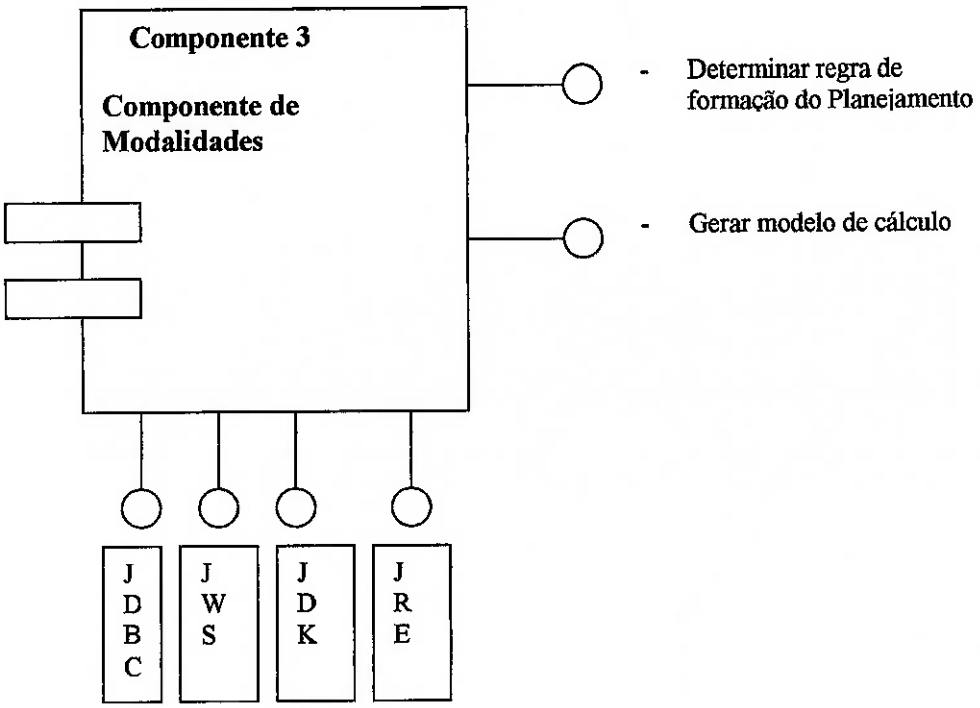
VLR_OPER_ID	Identificação do valor base. Chave primária.
OPER_FINANC_ID	Identificação da operação financeira em que o valor base está sendo utilizado. Chave estrangeira para a tabela OPER_FINANCEIRA.
TIPO_COMP_ID	Identificação do valor base. Exemplo: Valor da taxa de juros, valor da taxa de comissão, etc. Chave estrangeira para a tabela TIPOS_COMPONENTES.
VALOR_BASE_OPER	Valor informado pelo usuário para o valor base.
TIPO_FREQ_ID	Identificação do tipo de freqüência. Chave estrangeira para a tabela TIPO_FREQUENCIAS.

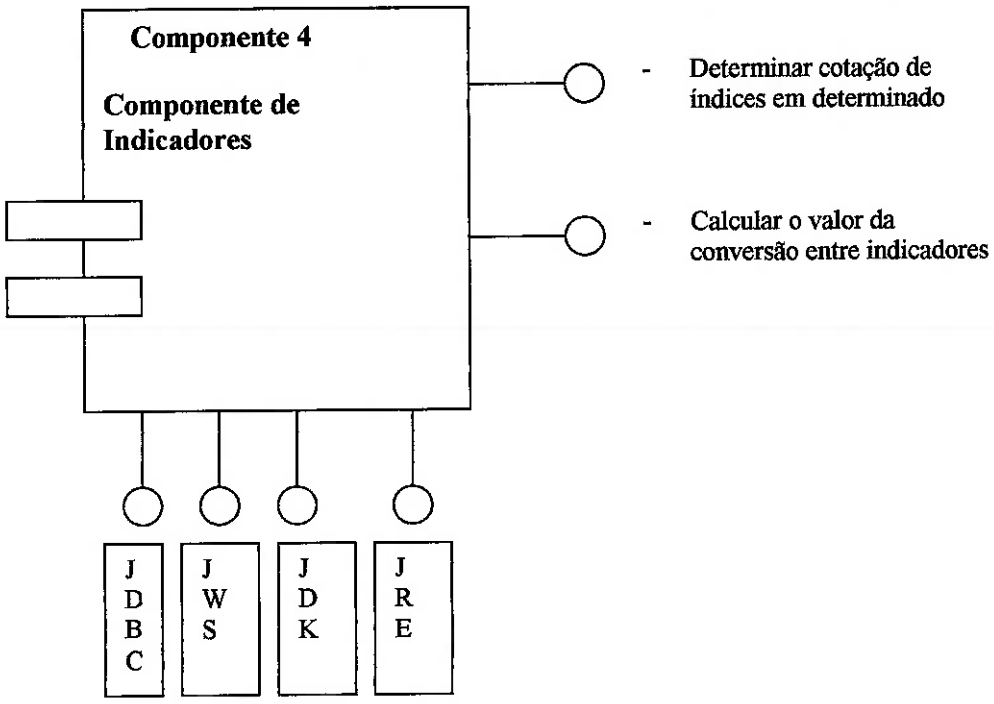
ANEXO I – DIAGRAMA DE COMPONENTES




Componente 1
Componente Principal
- Determinar regra de formação do Planejamento
- Gerar modelo de cálculo
JDBC
JWS
JDK
JRE
JSP


Componente 2
Componente de Cálculo
- Determinar condição válida
- Montar composição de condições
- Decompor fórmula
- Calcular componente de cálculo
JDBC
JWS
JDK
JRE


Componente 3
Componente de Modalidades
- Determinar regra de formação do Planejamento
- Gerar modelo de cálculo
JDBC
JWS
JDK
JRE


Componente 4
Componente de Indicadores
- Determinar cotação de índices em determinado
- Calcular o valor da conversão entre indicadores
JDBC
JWS
JDK
JRE

ANEXO J – INTERFACE HOMEM-MÁQUINA

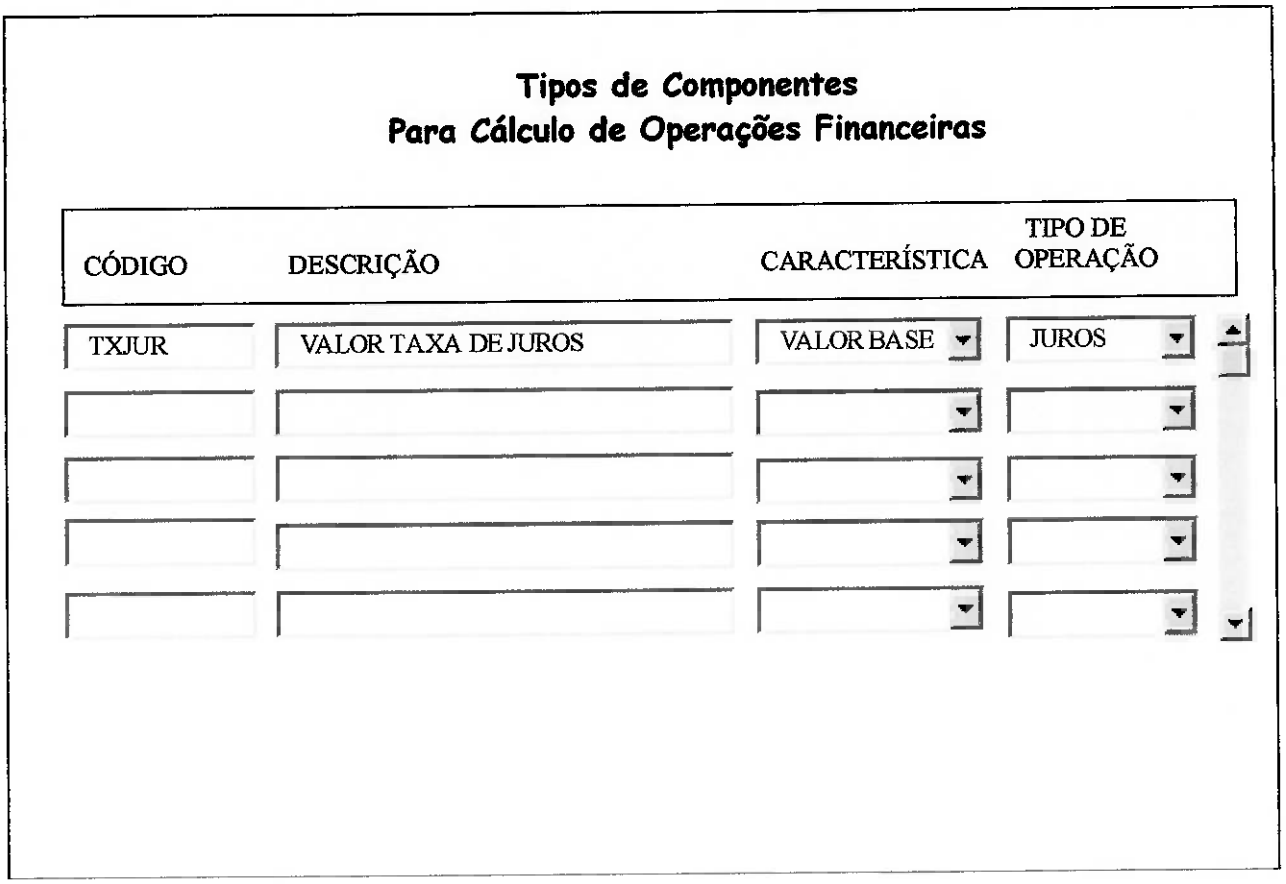
A seguir estão representadas as interfaces com o usuário que apresenta as seguintes características:

- Facilidade de Aprendizado e Facilidade de Memorização.
  - Usuários menos experientes e com necessidade de aprendizado rápido para produtividade.
    - A interface deve ser similar ao Excel para simular operações financeiras pois é a ferramenta mais utilizada pelo usuário
    - F1 padrão de help "sensivel:"ao contexto / localização do cursor.
    - Campos obrigatórios devem ter sua legenda apresentada em vermelho.

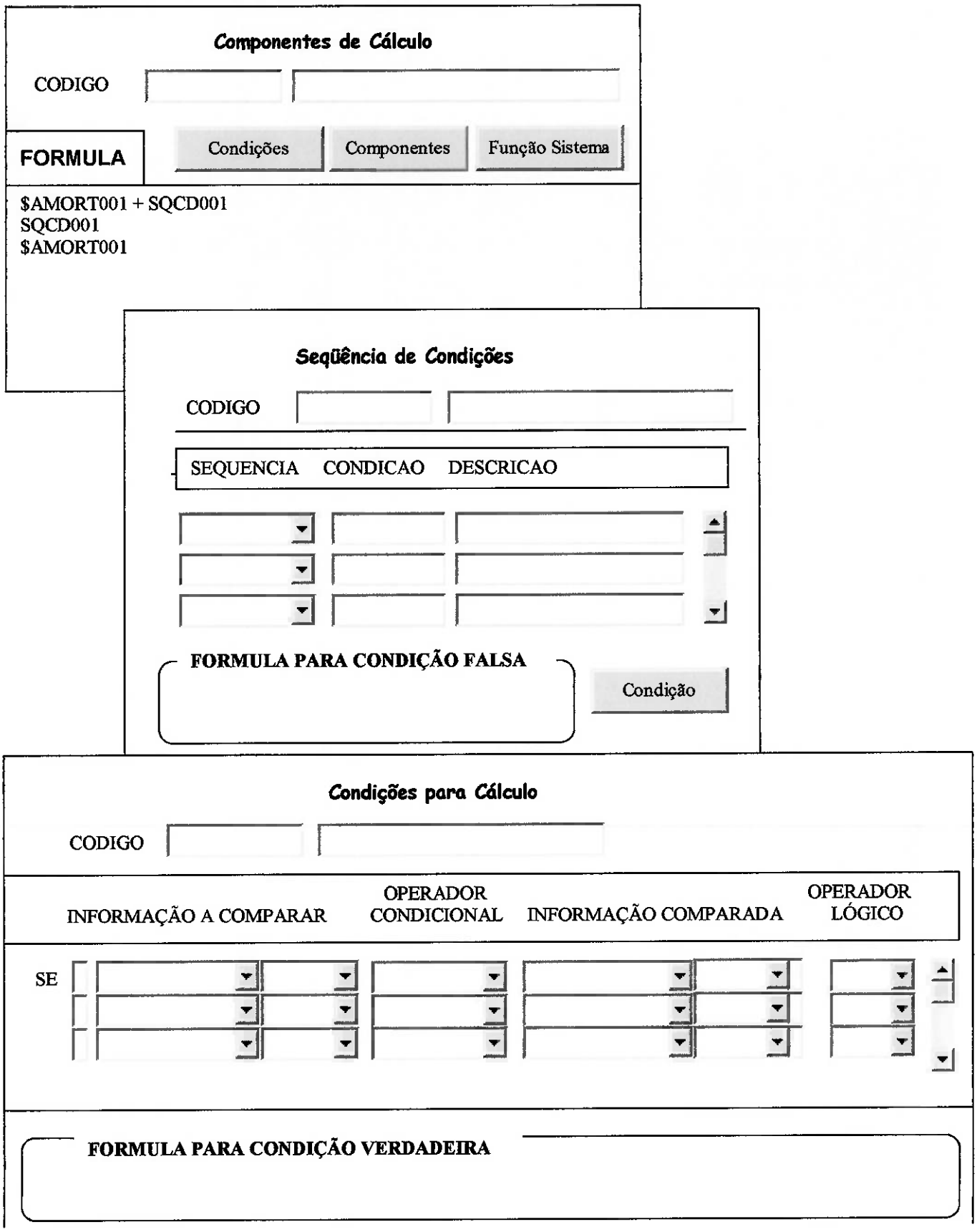
- Eficiência
  - Foco principal produtividade e agilidade no processamento de tarefas.
    - A tecla "enter" deve ser padronizada como mudança de campo ou tela.
    - A interface deve disponibilizar a qualquer momento a opção via botão de visulização de planejamento do planejamento, bem como recálculo da operação.

- Erros
  - E importante assegurar a exatidão dos processos e informações demonstradas visando evitar/minimizar erros de análise que podem levar a prejuízos operacionais da instituição.
    - Valores fora de padrões aceitáveis

1) Modelo de Interface para especificação de Tipos de Componentes para Cálculo



2) Modelo de Interface para determinar Componentes de Cálculo



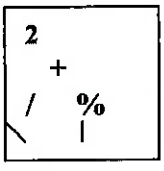
3) Modelo de Interface para Simulação de Operações Financeiras

Assistente para criação de Operação Financeira

VALORES BASE

CODIGO	DESCRICAO	VALOR	FREQUENCIA
	VALOR DO CONTRATO	10.000,00	MENSAL
TXJ	TAXA DE JUROS	2	

PARCELA				DATA / COMPONENTES	AMP01		PRS01		TXJ01		JUR02		JURAC		PRS04		SDV01		SDV21	
0	1	P1	D1	15-06-2003					E1	TXJ01	E1	JUR02	E1	JURAC			E1	SDV01		
0	1	P1	D2	30-06-2003															E1	SDV21
1	1	P2	D3	15-07-2003					E2	TXJ01	E2	JUR02	E2	JURAC			E2	SDV01		
1	1	P2	D4	17-07-2003	E1	AMP01	E1	PRS01							E1	PRS04				
1	1	P2	D5	31-07-2003															E2	SDV21
2	1	P3	D6	15-08-2003					E3	TXJ01	E3	JUR02	E3	JURAC			E3	SDV01		
2	1	P3	D7	31-08-2003															E3	SDV21
3	1	P4	D8	15-09-2003					E4	TXJ01	E4	JUR02	E4	JURAC			E4			
3	1	P5	D9	18-09-2003	E2	AMP01	E2	PRS01							E2	PRS04				
3	1	P5	D10	30-09-2003															E4	SDV21
4	1	P6	D11	15-10-2003					E5	TXJ01	E5	JUR02	E5	JURAC			E5	SDV01		
4	1	P7	D12	31-10-2003															E5	SDV21
5	1	P8	D13	15-11-2003					E6	TXJ01	E6	JUR02	E6	JURAC			E6	SDV01		
5	1	P8	D14	16-11-2003	E3	AMP01	E3	PRS01							E3	PRS04			E6	SDV21



LISTA DE REFERÊNCIAS

AMBLER,S.W. **Análise e projeto orientados a objeto**, 2.ed. Rio de Janeiro: Infobook, 1998.

BECERRA,J.L.R. Aplicabilidade do Padrão de Processamento Distribuído e Aberto nos Projetos de Sistemas Abertos de Automação, São Paulo. Dissertação (Doutorado) – Escola Politécnica, Universidade de São Paulo.

BEZERRA,E. **Princípios de análise e projeto de sistemas com UML**, 1.ed. Rio de Janeiro: Campus, 2002.

CHAMPEAUX, D.; ANDERSON, A.; FELDHOUSEN,E. **Case Study of Object-Oriented Software Development**, OOPSLA'92, Vancouver, British Columbia, Canada, p.377-391.

FURLAN, J.D. **Modelagem de Objetos através da UML – the Unified Modeling Language – Análise e desenho orientados a objeto**, 1.ed. São Paulo: Makron Books,1998.

JUNIOR, C.A . **Uma Nova Era na Tecnologia dos Bancos de Dados**, InterSystems Corporation, São Paulo, 2002.

MARTIN, J. **Princípios de análise e projeto baseados em objetos**, 6.ed. Rio de Janeiro:Campus,1994.

MELLO, R.; CHIARA,R.; VILLELA,R. **Aprendendo JAVA2**, 1.ed. São Paulo: Novatec Editora Ltda., 2002.

MELO, A.C.; **Desenvolvendo aplicações com UML**, 1.ed. Rio de Janeiro: Brasport, 2002.

PAGE-JONES,M. **Fundamentos do desenho orientado a objeto com UML**, 1.ed. São Paulo: Makron Books, 2001.

RUMBAUGH,J. et al. **Modelagem e projetos baseados em objetos**, 1.ed. Rio de Janeiro: Campus, 1994.

SILVA,A. **Dominando a tecnologia de objetos**, 1.ed. São Paulo: Editora BookExpress Ltda., 2002.

SINTES, T. **Aprenda programação orientada a objetos em 21 dias**, 1.ed. São Paulo: Pearson Education do Brasil, 2002.

TONSIG, S.L. **Engenharia de Software**, 1.ed. São Paulo: Futura, 2003.

ZIMBRÃO, G. **Mapeando um modelo de classes para um banco de dados relacional**. SQL Magazine, Rio de Janeiro, n.5, p.28-33, Ano 2003.