UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
MESTRADO PROFISSIONALIZANTE EM ENGENHARIA

UTILIZAÇÃO DE PADRÕES DE ANÁLISE NO DESENVOLVIMENTO DE MODELO PARA SISTEMAS DE GESTÃO EMPRESARIAL

Fernando Antônio Sonda

Porto Alegre, 2002

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
MESTRADO PROFISSIONALIZANTE EM ENGENHARIA

UTILIZAÇÃO DE PADRÕES DE ANÁLISE NO DESENVOLVIMENTO DE MODELO PARA SISTEMAS DE GESTÃO EMPRESARIAL

Fernando Antônio Sonda

Orientador: Professor Dr. Leonardo Rocha de Oliveira

Banca Examinadora:

Prof. Dr. Cláudio Walter

Prof. Dr. José Luis Duarte Ribeiro

Prof. Dr. Ricardo Melo Bastos

Trabalho de Conclusão do Curso de Mestrado Profissionalizante em Engenharia como requisito parcial à obtenção do título de Mestre em Engenharia – modalidade Profissionalizante – Ênfase Produção

Porto Alegre, 2002

Este Trabalho de Conclusão foi analisado e julgado adequado para a obtenção do título de mestre em ENGENHARIA e aprovada em sua forma final pelo orientador e pelo coordenador do Mestrado Profissionalizante em Engenharia, Escola de Engenharia, Universidade Federal do Rio Grande do Sul.

_____

Prof. Leonardo Rocha de Oliveira

Orientador
Escola de Engenharia
Universidade Federal do Rio Grande do Sul

_____

Prof[a]. Helena Beatriz Bettella Cybis

Coordenadora
Mestrado Profissionalizante em Engenharia
Escola de Engenharia
Universidade Federal do Rio Grande do Sul

BANCA EXAMINADORA

Prof. Cláudio Walter
PPGEP/UFRGS

Prof. José Luis Duarte Ribeiro
PPGEP/UFRGS

Prof. Ricardo Melo Bastos
PUC-RS

Dedicado a Olindo, Iracema e Silvia

AGRADECIMENTOS

A todos que acreditaram em mim e tornaram possível este trabalho.

ÍNDICE

LISTA DE FIGURAS

RESUMO

Este trabalho apresenta um modelo genérico para o desenvolvimento de sistemas de gestão empresarial. O modelo foi criado com base na experiência profissional do autor do trabalho e revisão bibliográfica sobre a utilização de padrões de análise. O modelo proposto prioriza aspectos de simplicidade e flexibilidade no desenvolvimento de sistemas de informação. Sua utilização visa facilitar a reutilização de componentes, diminuir o tempo para o desenvolvimento de aplicativos e propiciar a criação de sistemas flexíveis que rapidamente possam se adaptar a novas necessidades dos processos de negócios. Os modelos estão descritos no corpo do trabalho e estendem a utilização dos padrões originais de onde os mesmos se originam, descrevendo os relacionamentos entre os modelos apresentados, suas classes e atributos. Exemplos da utilização prática dos modelos são apresentados em situações de negócios e conclusões sobre o potencial de utilização dos mesmos são discutidos ao final do trabalho.

ABSTRACT

This work presents a generic model to help building information systems for business management. It is a result from the author's professional experience and a bibliographical review on analysis patterns utilization. The proposed model aims at providing system's simplification and flexibility, making easier reutilization of its components, reducing application development time and allowing systems that could quickly adapt to new business needs. Details of the proposed models are presented in this work and provide examples to use facilitate further use of the original patterns, and showing how models connect themselves, as well as their classes and attributes. The examples are applied to practical business situations and conclusions are drawn.

GLOSSÁRIO

Agregação (*aggregation*) - "É uma forma especial de associação que especifica o relacionamento "parte-todo" ou "uma-parte-de" no qual os objetos que representam os componentes são associados a um objeto que representa a estrutura inteira" (RUMBAUGH; et al., 1994).

Associação (*association*) - "Relacionamento estruturado que descreve um conjunto de ligações na qual uma ligação é uma conexão entre objetos (BOOCH et al., 1999).

Atributo[1] (*attribute*) - "Propriedade nomeada de um classificador que descreve um conjunto de valores que as instancias da propriedade podem conter" (BOOCH et al., 1999). "Uma característica do classificador que descreve o conjunto de valores que as instancias do classificador podem conter" (ERIKSSON; PENKER, 2000). "Atributos do ponto de vista conceitual são associações" (FOWLER; SCOTT, 2000).

Banco de dados relacional - "Banco de dados que é percebido pelo usuário como um grupo de tabelas" (DATE, 1989)

Classe[2] (*class*) - "É a descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica" (BOOCH et al., 1999).

Classificador (*classifier*) - "Mecanismo que descreve as características estruturais e comportamentais. São classificadores as classes, interfaces, tipos de dados, sinais, componentes, nodos, casos de uso e sub-sitemas" (BOOCH et al., 1999).

Coluna (*column*) - No modelo relacional é utilizado o termo coluna sendo este equivalente a atributo (MULLER, 1999)

Estereótipo (*stereotype*)- "Extensão do vocabulário da UML, na qual habilita a criação de novos blocos de construção que são derivados dos já existentes mas são específicos ao problema" (BOOCH et al., 1999).

Generalização (*generalization*) - "Relacionamento entre uma classe e uma ou mais versões refinadas dela" (RUMBAUGH; et al., 1994).

Instância (*instance*)-"Membro individual de um tipo ou classe" (LARMAN, 1998)

---

[1] No texto os atributos, relacionamentos e colunas estão em negrito com a primeira letra das palavras em maiúscula com acentuação (os modelos não possuem acentuação). Ex: RazãoSocial**.**

[2] No texto, deste trabalho, as classes e tabelas estão escritas com letras maiúsculas, com acentos, podendo estar no plural ou singular (nos modelos sempre estão no singular e sem acentuação) para uma melhor compreensão do texto. Ex: RECURSO.

Modelo (*model*) - "Descrição das características estáticas ou dinâmicas sobre uma determinada área, retratada através de diferentes pontos de vista" (LARMAN, 1998)

Objeto[3] (*object*) - "Manifestação concreta de uma abstração; uma entidade que encasula estado e comportamento; uma instância de uma classe" (BOOCH et al., 1999). "É simplesmente alguma coisa que faz sentido no contexto de uma aplicação; algo com limites nítidos e significado em relação ao problema em causa" (RUMBAUGH; et al., 1994).

Padrão (*pattern*) - "Solução comum, para um problema, em um contexto específico" (BOOCH et al., 1999).

Propriedade (*propriety*) - "Valor nomeado que descreve uma característica de um elemento" (BOOCH et al., 1999).

Relacionamento (*relationship*) - "É uma conexão entre coisas"; Associação, generalização e dependências são tipos de Relacionamento (BOOCH et al., 1999).

Regras de negócio (*Business Role*) – Declaração que define ou restringe aspectos do negócio. É destinado para afirmar as estruturas dos negócios ou controlar ou influenciar o comportamento do mesmo" (HALLE, 2001).

String - Seqüência de caracteres de texto (BOOCH et al., 1999).

Super Tipo (*Power Type)*- "Especifica um classificador da qual seus objetos são filhos de um determinado pai" (BOOCH et al., 1999).

Template - Elemento parametrizado (BOOCH et al., 1999).

Tabela (*table*) - Grupo não ordenado de linhas

Tipo[4] (*type*)- "Estereótipo de classe usado para especificar um domínio de objetos, junto com as operações aplicáveis aos objetos (BOOCH et al., 1999).

Framework - Arquitetura padrão que proporciona um molde (*template*) extensível para aplicações em um determinado domínio (BOOCH et al., 1999).

---

[3] No texto os objetos estão escritos entre apóstrofes. Ex: 'João' .

[4] No texto os tipos estão escritos com letras maiúsculas e em negrito.

1 INTRODUÇÃO

1.1 Visão Geral

Padrões de análise são modelos criados a partir da observação de diversos sistemas, refletindo a experiência prática de profissionais que contribuíram para criar modelos simples, genéricos e flexíveis.

Uma das dificuldades para o entendimento e a utilização prática dos padrões de análise, propostos principalmente por FOWLER (1996) e HAY (1996), se dá pela necessidade dos profissionais de informática de conhecer o funcionamento detalhado dos sistemas empresariais e ter capacidade para criar modelos detalhados, a partir de padrões de análise genéricos e sem detalhamento da sua utilização prática.

Este trabalho é o resultado da experiência profissional no desenvolvimento de sistemas empresariais, com a utilização dos padrões de análise. Objetiva-se criar um modelo genérico capaz de se adaptar às mais diversas formas de negócio e que possa se adequar às novas necessidades empresariais. Os modelos aqui propostos são apresentados de forma detalhada, estendendo os padrões originais dos quais os mesmos se originam e procurando mostrar como foram utilizados em situações práticas de negócio.

1.2 Objetivos

1.2.1 Objetivos Gerais

Este trabalho tem como objetivo a utilização de padrões de análise (*analysis patterns*) no desenvolvimento de modelo de sistema de informação para gestão empresarial.

1.2.2 Objetivos Específicos

- Apresentar revisão bibliográfica do estado da arte na utilização corrente de padrões de análise.

- Facilitar o desenvolvimento de novos sistemas fazendo com que os mesmos não sejam iniciados do zero.
- Propor um modelo flexível o suficiente que possa se adaptar a diferentes tipos de negócio.
- Facilitar a adequação dos sistemas às novas demandas do negócio.
- Facilitar o entendimento e utilização do modelo proposto pela sua simplicidade.
- Reduzir custos com desenvolvimento e manutenção do software.
- Facilitar a criação de componentes de software reutilizáveis.

A utilização de padrões de análise é o início de um trabalho mais amplo que objetiva a utilização de padrões em todas as fases do desenvolvimento de software, objetivando atender seus clientes com padrão de qualidade superior as expectativas.

1.3 Metodologia

A definição de uma metodologia em estágio anterior ao desenvolvimento dos modelos tem a vantagem de fornecer linhas gerais para organizar idéias e sugerir um curso de ação a ser seguido. O desenvolvimento de modelos, contemplando necessidades da empresa criadora do software, de seus clientes e de necessidades operacionais e administrativas dos processos empresariais são necessidades que devem ser consideradas na decisão pelo método de pesquisa que deve orientar o trabalho de dissertação.

Para o trabalho foi feita a opção pelo método de pesquisa conhecido como Prototipagem (Crinnion, 1992), o qual providencia linhas gerais para o desenvolvimento de sistemas de informação. A escolha pelo método deve-se ao fato deste estar baseado na criação de modelos que interativamente vão sendo melhorados ao longo das etapas de desenvolvimento do sistema. Como o protótipo pode eventualmente se tornar a versão final do sistema, os modelos intermediários tornam-se documentos do processo de criação do sistema.

A principal justificativa para a escolha do método de prototipagem está no fato dele permitir que a experiência dos profissionais envolvidos a criação do sistema seja

alinhada com as necessidades empresariais expressas pelos profissionais de negócios, os quais viriam a tornar-se os usuários da versão final do sistema.

A utilização do método de prototipagem permitiu a utilização de padrões de análise apresentados em referências bibliográficas como ponto de partida para a criação dos modelos propostos. A versão final dos modelos, descrita no Capítulo 4, representa uma evolução do estado original dos padrões de análise adaptados aos processos de negócios das empresas onde os sistemas foram implantados.

O modelo proposto é resultado da união entre a experiência dos profissionais da criação de sistemas com pesquisa bibliográfica sobre padrões, foi implementado e testado em situações de trabalho. Os sistemas foram criados utilizando-se bases de dados relacionais em arquitetura cliente-servidor e interfaces gráficas para usuários em programação orientada a objetos. Cabe salientar que o modelo proposto foi implementado em sistemas corporativos reais estando os mesmos funcionando em modo de produção.

1.4 Limitações

Este trabalho não tem o objetivo de detalhar o projeto e implementação do modelo proposto em um sistema real, mas propor um modelo que sirva de base para tanto.

O modelo proposto se adapta melhor às atividades de controle empresarial do que às de planejamento empresarial. As atividades de planejamento empresarial que envolvem simulações de cenários normalmente necessitam de modelos específicos que estendem o modelo proposto.

O modelo não tem a pretensão de ser um modelo “universal” que resolva todos os problemas de modelagem de negócios, mas que sirva de base para a implementação de sistemas de gestão empresarial e, conforme a necessidade, poderá ser estendido para melhor adequação às necessidades específicas dos negócios.

1.5 Organização do trabalho

O primeiro capítulo apresenta á visão geral do trabalho bem como seus objetivos, limitações e metodologia.

O segundo capítulo compara a indústria automobilística e a de informática enfatizando como foi importante a utilização dos padrões no desenvolvimento da indústria automobilística e enfatizando quanto a utilização dos mesmos pode ser útil nas empresas de informática

O terceiro capítulo proporciona uma visão rápida da "Unified Modeling Language" (UML) e aborda os benefícios da adoção da mesma.

O quarto capítulo apresenta a fundamentação teórica dos padrões de análise utilizados, seguida do modelo prático construído com base nos mesmos. Optou-se por apresentar desta forma para que o leitor possa ter um melhor entendimento dos modelos como também do relacionamento e evolução dos mesmos. Os modelos propostos são os que seguem:

Empresas e Usuários: Modelam as unidades empresariais que farão uso do sistema e seus usuários, determinando como os dados serão compartilhados entre as unidades empresariais e a estrutura de controle de acesso dos usuários aos mesmos.

Histórico (*log*): Modela a forma como os eventos do sistema serão monitorados e armazenados para uma eventual auditoria do sistema.

Unidades: Modela as unidades de medidas e monetárias juntamente com as taxas de conversão entre as mesmas.

Recursos: Modela em uma estrutura hierárquica múltipla todos os objetos que são relacionados pelos *Documentos*. Os objetos: produtos, clientes, contas, médicos, convênios, fornecedores, empregados, unidades organizacionais, entre outros, são exemplos de *Recursos* em nosso modelo.

Caráter: Este modelo é responsável por caracterizar os recursos e os documentos, ou seja, as propriedades de um recurso, do tipo pessoa, tal como peso, altura, idade entre outras são armazenadas neste modelo.

Documentos: Modela os dados contidos nos mais diversos tipos de documentos de uma organização como também o inter-relacionamento entre os mesmos.

O quinto capítulo é dedicado a sugestões de trabalhos futuros e conclusão final.

2 PADRÕES

Como a engenharia de software é uma ciência nova, em relação à engenharia civil ou mecânica, neste capítulo será descrita a evolução da indústria automobilística, como foco de aplicação de sistemas de informação, salientando-se a importância do uso de padrões como um dos elementos principais desta evolução. Através desta análise serão descritos os benefícios que os padrões podem proporcionar para a evolução das empresas de software.

2.1 Situação atual das empresas de software

O poder computacional e das redes de computadores cresceu nos últimos anos, mas os projetos de software continuam caros e com erros (FAYAD et al., 1999). Muito do custo e do esforço está relacionado ao contínuo redescobrimento e reinvenção de conceitos básicos. O crescimento heterogêneo do hardware, a diversidade de sistemas operacionais e de plataformas de comunicação dificulta a construção de sistemas corretos, portáteis, eficientes e baratos, tornando difícil a reutilização dos projetos, algoritmos e interfaces já existentes (FAYAD et al., 1999).

Conforme Johnson (apud BROWN et al., 1998) para cada seis projetos de software cinco não são considerados bem sucedidos e aproximadamente três são cancelados. O restante não cumpre o orçamento nem prazos originalmente planejados, além de, virtualmente, nenhum software ser capaz de acomodar mudanças dos negócios, sendo gasta a metade do custo do software em alterações e extensões ao mesmo (HOROWITZ apud BROWN et al., 1998). Empresas de informática, ao venderem um novo sistema, possuem uma expectativa de faturamento quatro vezes maior do que o valor base da negociação inicial, com treinamento, consultoria, suporte e manutenção, ou seja, quanto mais problemas o usuário tiver, mais as mesmas irão faturar (BROWN et al., 1998).

Conforme pesquisa realizada em 1994, pelo Instituto de Engenharia de Software da Universidade de Carnegie Mellon, 70% das empresas de informática estão no nível CMM 1 (Capability Maturity Model 1) (JACOBSON et al., 1997). As

empresas no nível um se caracterizam por não possuírem procedimentos formalizados, estimativas de custos e nem planejamento de projeto (JACOBSON et al., 1997). Portanto, o processo de desenvolvimento de software é personalizado, às vezes caótico e dependente do esforço individual das pessoas (JÉZÉQUEL et al., 2000).

Devido a aspectos particulares da engenharia de software, empresas de informática, atualmente, trabalham de forma similar aos artesões medievais, os quais esculpiam suas gárgulas escondidos, sem revelar suas técnicas de trabalho (FOOTE, 1998).

2.2 Necessidades por padrões na empresa Ph.D. Informática

A Ph.D. Informática é uma empresa de desenvolvimento de software (*softwarehouse*) situada em Caxias do Sul, um dos principais pólos industriais do estado do Rio Grande do Sul. Sua principal atividade é desenvolver soluções personalizadas às necessidades dos seus clientes e assessora-los no uso de tecnologias de informática. A empresa vem desenvolvendo sistemas para atender empresas regionais atuando principalmente na área de saúde e industrial.

A Ph.D. Informática sempre optou por estar tecnologicamente atualizada em relação as mais novas tecnologias de desenvolvimento de software do mercado. Esta abordagem associada a sua característica de desenvolver sistemas personalizados às necessidades de seus clientes faz com que seus novos projetos tenham que ser desenvolvidos do zero.

Ao desenvolver os mais variados tipos de sistemas a Ph.D. Informática observou que poderia criar um modelo base, para atender as necessidades de seus clientes, apesar dos mesmos possuírem atividades diferentes.

Com intuito de melhorar o processo de desenvolvimento de software, a Ph.D. resolveu buscar na bibliografia existente soluções para minimizar este problema. Nesta busca concluiu que não era a única empresa que possuía este problema. A revisão bibliográfica apresentada no capítulo dois demonstra que este é um problema comum a empresas de desenvolvimento de software.

Como a Ph.D. informática sempre esteve ligada ao desenvolvimento de sistemas industriais achou conveniente se inspirar na evolução do processo produtivo das industrias, principalmente nos conceitos de padronização, simplificação e flexibilidade. Assim sendo, o trabalho de pesquisa indicou que a solução de seus problemas passavam necessariamente pela padronização no seu processo de desenvolvimento de software.

FAYAD et al. (1999) afirmam que ao desenvolver sistemas é desejável que padrões sejam aplicados desde as fases iniciais do ciclo de vida dos mesmos sendo que quanto antes forem utilizados, maiores serão os ganhos proporcionados pelos mesmos.

Desta forma a Ph.D.Informática focou seu esforço inicial na melhoria da qualidade de seus sistemas, desenvolvendo padrões de análise.

2.3 O uso de padrões nas empresas automotivas

O ramo das empresas automobilísticas atualmente se caracteriza por ser um ramo com forte concorrência globalizada, o qual obriga as empresas a uma constante atualização tecnológica em seus processos e produtos, as quais, com o passar dos anos, estão produzindo carros com maior qualidade e menor custos. Ao analisar as mudanças tecnológicas que proporcionaram o avanço deste setor se salienta o impacto da padronização na evolução da mesma e se traça um paralelo do setor automobilístico com o de software salientando os benefícios que este pode vir a ter com o uso maior da padronização.

Segundo WOMACK et al. (1997), as empresas artesanais fabricantes de automóveis, do início do século, se caracterizavam por:

a) Força de trabalho qualificada com habilidades artesanais
b) Os empregados podiam esperar ter seu próprio negócio
c) Organizações descentralizadas sendo abastecidas por pequenas empresas
d) O proprietário administrava seu negócio com contato direto com consumidores, fornecedores e empregados.
e) Baixo volume de produção, com produção personalizada ao gosto do Cliente

f) Produtos sem diferenciais competitivos não possibilitando a formação de monopólios.

Segundo WOMACK et al. (1997), este tipo de produção apresentava as seguintes desvantagens:

a) Não haviam ganhos de escala.
b) Os produtos não eram confiáveis pois todo o produto era um protótipo
c) Não havia a garantia da qualidade do produto por falta de testes sistemáticos
d) Não possuíam pesquisas sistêmicas impedindo assim as inovações fundamentais

O surgimento da produção em massa de Henry Ford veio a superar problemas inerentes à produção artesanal. Em 1908, o Modelo T da Ford conseguiu atingir dois objetivos: (i) produzir um carro projetado para a manufatura e (ii) ser de fácil utilização pelo usuário ("user-friendly"), ou seja, qualquer pessoa poderia ser capaz de dirigir ou consertar o carro sem precisar de seu próprio motorista ou mecânico (WOMACK et al., 1997).

A chave para a produção em massa não residia na linha de montagem em movimento contínuo, mas na completa e consistente intercambialidade das peças e facilidade de ajustá-las entre si. Para isto Ford insistiu na padronização das medidas dos componentes, percebendo os benefícios financeiros que resultariam. Nenhuma outra indústria da época percebeu a relação de causa e efeito e perseguiu a padronização com o fervor quase religioso de Ford (WOMACK et al., 1997).

A intercambialidade, simplicidade e facilidade de ajustes proporcionaram a Ford uma vantagem competitiva, permitindo até eliminar ajustadores qualificados, que constituíam o maior contingente da força de trabalho da época (WOMACK et al., 1997).

No início dos anos 20, com a entrada de Alfred Sloan na General Motors, o mesmo vislumbrou a necessidade de oferecer modelos diferentes para mercados diferentes. Assim sendo criou cinco novos modelos em ordem crescente de preço, do

Chevrolet ao Cadillac, atendendo compradores potenciais de todas as rendas (WOMACK et al., 1997).

Sloan conseguiu resolver o conflito entre a necessidade de padronização e a diversidade de modelos exigida pela demanda variada dos consumidores. Padronizou vários itens mecânicos em toda a faixa de produtos da companhia e alterava anualmente apenas a aparência externa dos carros, introduzindo uma série de novos acessórios para manter o interesse dos consumidores (WOMACK et al., 1997).

Após a segunda guerra mundial o Japão surgiu como uma nova potência na indústria automobilística, Taiichi Ohno da Toyota, agressivamente perseguiu a eliminação das perdas dos produtos e processos, estratégia esta que levou a Toyota a produzir produtos com alta qualidade, baixos custos e curto tempo de passagem ("lead-time") (POPPENDIECK, 2001).

Em 2000 em Gravataí, Rio Grande do Sul, Brasil foi instalada pela GM a fábrica de automóveis mais moderna do mundo da atualidade. O diferencial desta nova fábrica, em relação a seus concorrentes é (OGAWA, 2001):

a) A utilização do conceito de sistemistas, que se instalam ao lado da montadora.
b) A simplificação do produto, o qual possui a metade das peças de um carro convencional.
c) A simplificação do processo que unificou até sete etapas em uma única.
d) A ligação direta da produção da fábrica com a internet, o cliente pode escolher os opcionais do seu carro, de sua casa.

Com esta breve história da evolução da indústria automobilística, queremos deixar claro que os padrões associados à simplificação do produto e processos proporcionaram a passagem da fase artesanal para a industrializada e, ainda hoje, continuam tendo papel fundamental para a redução dos custos, personalização do produto, redução do tempo de entrega e melhoria da qualidade, fatores competitivos fundamentais para as empresas entrarem no mercado eletrônico atual(*e-bussines*), fortemente estruturadas para uma competição globalizada nunca vista anteriormente.

2.4 Histórico dos padrões

O uso de padrões na indústria foi um dos fatores que marcou a revolução industrial do início do século (WOMACK et al., 1997). Em 1979 o arquiteto Christopher Alexander em seu livro "The Timeless Way of Building" lançou as noções dos padrões em arquitetura descrevendo os mesmos como "a qualidade sem nome" (RISING, 1998). Em 1995, Erich Gamma, Richard Helm, Ralph Johnson, e John Vlissides, conhecidos como a gang dos quatro (*Gang of Four, GoF*) (BUSCHMAN et al., 1996), lançaram o livro "Design Patterns: Elements of Reusable Object-Oriented Software" (GAMMA et al., 1995) onde apresentaram o primeiro catálogo de padrões de projetos orientados a objetos (BUSCHMAN et al., 1996). Este livro se tornou um referencial na área de informática relativo ao uso de padrões. Desde então, livros têm sido lançados sobre o assunto, vários fóruns se desenvolveram onde programadores, professores, práticos e teóricos anunciam e discutem seus padrões (FOOTE, 1998). Desta forma pode-se desenvolver sistemas aprendendo e evoluindo rapidamente, com a utilização de padrões utilizados e divulgados pelos peritos da área (COAD et al., 1995).

2.5 A busca de padrões e componentes reutilizáveis em software

A indústria de software tem buscado formas que permitam a reutilização de códigos há anos. Primeiro com a análise e programação estruturada e, mais recentemente, com tecnologias orientadas a objetos. Uma das promessas das tecnologias orientadas a objeto é a possibilidade de compor um novo software a partir da união de componentes de fornecedores diversos. Dentro desta perspectiva, a tarefa de criação de software se tornaria similar à montagem de um brinquedo tipo Lego, para o qual é necessário apenas comprar as partes e uni-las para formar o brinquedo. Até hoje esta promessa não se realizou. Sistemas empresariais são complicados, construir uma biblioteca de objetos reutilizáveis continua sendo o sonho de desenvolvedores de software (FOWLER, 2000b). Pesquisa realizada por Capers Jones da SPR (Software Productivity Research Inc.), indica que apenas 15% do código da

linguagem C são reutilizados, acontecendo o mesmo com as demais linguagens (SUTHERLAND, 1997).

Flexibilidade, velocidade, qualidade e baixo custo estão diretamente relacionados à reutilização de objetos de negócio. No entanto, a reutilização de objetos é difícil, pois exige a procura por objetos reutilizáveis e o entendimento dos mesmos pode custar mais do que construir um novo objeto (SUTHERLAND, 1997).

FAYAD et al. (1999) afirma que a reutilização ao nível de projeto é mais importante do que a de código, porque pode ser aplicada em contextos diferentes mais freqüentemente. O fato de aplicar padrões o quanto antes no processo de desenvolvimento de software proporciona um maior impacto na aplicação como um todo (FAYAD et al., 1999). Padrões de análise, projeto e código são importantes, mas a longo prazo, padrões de análise e projeto são os que proporcionam os melhores resultados (BIGGERSTAFF apud FAYAD et al., 1999).

A reutilização de objetos na engenharia de software pode ser comparada a intercambialidade das peças na indústria automotiva. Quando as indústrias automotivas ainda eram artesanais, as peças tinham que ser adaptadas ou ainda personalizadas para cada veículo, somente após a busca incessante da padronização por Ford é que a indústria automotiva conseguiu utilizar a mesma peca em carros diferentes sem a necessidade de "personalização" da mesma para cada cliente. Para usufruir a capacidade de reutilização de componentes, que permitiriam a montagem de sistemas como se fossem carros da indústria automotiva, é necessária uma infraestrutura padrão de objetos, capaz de suportar componentes reutilizáveis (SUTHERLAND, 1997).

2.6 Definição de Padrões

Os autores ALEXANDER, FOOTE, FOWLER e ERIKSON definem padrões da seguinte forma:

> "Cada padrão descreve um problema recorrente e a parte central da solução do problema, de forma que se possa usar esta solução repetidamente, sem utilizá-la da mesma forma por duas vezes" (LINDA apud ALEXANDER, 1977, p.x.).

> "Padrões são soluções repetitivas desenhadas pela experiência, não são invenções" (FOOTE, 1998).
>
> "Um padrão é uma idéia útil em um contexto prático e que provavelmente será útil em outro" (FOWLER, 1997).
>
> "Padrões são soluções genéricas estabelecidas que resolvem problemas que são comuns em diferentes situações de negócio. Eles podem ser reutilizados repetidamente e podem ser combinados e adaptados de várias formas diferentes" (ERIKSSON; PENKER, 2000).

Segundo BUSCHMANN et al. (1996), os padrões se caracterizam por:

a) Objetivam solucionar problemas recorrentes que surgem em situações específicas e apresentam a solução ao problema.
b) Documentam experiências existentes e bem provadas, mas não são invenções criadas artificialmente.
c) Proporcionam vocabulário comum e entendimento dos princípios do sistema.
d) São meios de documentar a arquitetura do software, evitando a violação da visão básica do software.
e) Proporcionam um esqueleto funcional base que ajudará na implementação da aplicação.
f) Ajudam a construir softwares de arquiteturas complexas e heterogêneas.
g) Ajudam a administrar a complexidade do software.

LINDA (1998) afirma que: "...é papel fundamental dos padrões capturar as melhores idéias de nossas experiências coletivas de forma a possibilitar o compartilhamento com outras pessoas da comunidade. Sua simplicidade é profunda e permitem compartilhar nossas melhores idéias com outros. Como resultado, todos evoluímos e nossas melhores idéias coletivas se tornarão melhores".

Padrões de análise baseiam-se no pré-suposto básico de que estruturas empresariais são similares. Entender as similaridades dos sistemas possibilita ao analista ter um modelo inicial que pode ser ajustado as necessidades específicas de uma empresa em particular. Padrões não têm a intenção de criar sistemas iguais para

todas as empresas. Pelo contrário, é reconhecido que todo o sistema tem suas particularidades e que as mesmas devem ser atendidas. Por outro lado, também é reconhecido que empresas com negócios diferentes, tais como governos, indústrias e serviços, possuem componentes similares (HAY, 1996). Muitas vezes, na modelagem de sistemas distintos, são observadas estruturas idênticas ou com atributos e relacionamentos muito parecidos. Estas observações possibilitam ao analista criar modelos simples e genéricos, que atendam os conceitos prévios descobertos e sejam capazes de sugerir novas idéias (HAY, 1996).

Atualmente, há modelos que retratam uma forma particular de negócio, não se preocupando com a sua essência. Padrões não são usados para normatizar o sistema de notação, mas como forma de pensar sobre negócios. Empresas, por mais diferentes que possam parecer, possuem estruturas similares. O entendimento destas similaridades permite ao analista um modelo base, o qual pode ser ajustado às necessidades de circunstâncias específicas (HAY, 1996). FOWLER (1997) afirma que sistemas empresariais não devem ser organizados seguindo a linha tradicional dos negócios, mas sim através de conceitos abstratos de processos que servirão de base para criação de modelos estruturais do sistema.

2.7 Vantagens do uso de padrões

”Projetos falham, mesmo aqueles que utilizam a última tecnologia, por falta de soluções simples” (CUNNINGHAM apud FOWLER, 1997). Ao modelar sistemas, decisões devem ser tomadas e não são certas ou erradas, mas sim, mais ou menos úteis (FOWLER, 1997). As decisões tomadas na modelagem afetam a flexibilidade e reusabilidade do sistema resultante. Construir um sistema muito flexível pode torná-lo complexo, sendo isto má engenharia (FOWLER, 1997). Achar uma solução simples para um problema toma tempo, esforço e pode ser frustrante. Quando achamos esta solução, a mesma parece óbvia e pode não ficar claro se foi entendido o real motivo que justifica o tempo necessário para se chegar à mesma (FOWLER, 1997).

“Modelos simples sempre recompensam o esforço” (FOWLER, 1997). Não somente fazem as coisas mais fáceis de serem construídas, mas o mais importante,

facilitam a manutenção e expansão futura do sistema (FOWLER, 1997). Uma melhor estruturação do software e a redução da complexidade fazem o software ser de fácil leitura e entendimento, facilitando a sua expansão e reusabilidade (PREE, 1995). Criar modelos simples que resolvam problemas complexos, por mais paradoxal que possa parecer, é uma tarefa complexa (FOWLER, 1997). O sistema contábil é um exemplo de modelo simples. Criado por um pastor enquanto contava as suas ovelhas há centenas de anos, este modelo sobrevive até hoje pela sua simplicidade (FOWLER, 2000c).

Seja reduzindo a complexidade dos sistemas ou nomeando e definindo abstrações, os padrões criam um vocabulário comum na equipe de desenvolvimento (JÉZÉQUEL et al., 2000). Os nomes dos padrões formam um vocabulário que auxilia os desenvolvedores a se comunicarem melhor (VLISSIDES, 1998), possibilitando que os mesmos discutam os problemas e as soluções de maneira efetiva e rápida (BUSCHMANN et al., 1996).

Padrões capturam o conhecimento de profissionais experientes, auxiliando no compartilhamento do conhecimento entre os desenvolvedores. A utilização de padrões também pode ser vista como ferramenta de aprendizado, mostrando boas técnicas de modelagem e desenvolvimento (ERIKSSON; PENKER, 2000). Quando da entrada de novos profissionais de informática na equipe de desenvolvimento, estes deverão entender os padrões existentes e, desta forma, absorver as experiências anteriores, para compreender a estrutura dos sistemas existentes, tornando-se aptos a criarem novos sistemas dentro dos padrões em uso. Com a utilização de padrões, a empresa cria um entendimento uniforme de seus sistemas, sendo que analistas do grupo possuirão o conhecimento básico sobre a modelagem dos processos, estando aptos a alterarem os mesmos em pouco tempo. Com isso, o projeto deixa de ser dependente de uma única pessoa, a qual não transmite seus padrões e suas técnicas.

Padrões podem exigir mais trabalho que soluções personalizadas. Porém, o esforço adicional invariavelmente pode trazer dividendos na forma de maior flexibilidade (GAMMA et al., 1995). Por ter estrutura básica genérica, o sistema se

adapta às novas demandas dos usuários rapidamente, ou até mesmo a uma mudança completa do negócio da empresa (HAY, 1996).

O aumento da velocidade no desenvolvimento de sistemas é uma das razões mais importantes para se trabalhar com padrões (BUSCHMANN et al., 1996). Ao começarmos um novo sistema, ou módulo, não se começa do zero. Este fato faz com que se tenha uma velocidade maior no desenvolvimento das aplicações.

Uma das maneiras usadas para medir a qualidade de um sistema é avaliar se os desenvolvedores seguiram os padrões. Focalizar o uso de padrões em um sistema pode levar a uma arquitetura menor, mais simples e muito mais compreensível do que aquelas produzidas sem padrões (GAMMA et al., 1995).

Apesar das vantagens citadas para o uso de padrões, os mesmos não são solução para todos os problemas de desenvolvimento de software. Padrões por si só não garantem alta produtividade ou flexibilidade. Os mesmos não eliminam a necessidade de pessoas criativas no processo produtivo capazes de fazer uso efetivo dos mesmos. Portanto, padrões constituem-se em mais uma ferramenta que os profissionais podem utilizar para o desenvolvimento de sistemas (VLISSIDES, 1998).

2.8 A necessidade de utilizarmos modelos para expressarmos os padrões

Desenvolvimento de software é uma engenharia, cujo processo de aprendizado pode ser catalisado pelo fato de poder olhar e consultar outras construções. Procurar acertos de outros engenheiros, entender os problemas e visualizar se estes foram ou não resolvidos são aspectos relacionados ao aprendizado humano. No caso de software, um dos problemas está no fato das estruturas de programação não estarem aparentes como as estruturas da construção civil ou de máquinas de manufatura (FOWLER, 2000b). Desta forma, faz-se necessário o uso de modelos para podermos comunicar estas estruturas e é necessário que seus autores as divulguem.

A modelagem é a parte central do desenvolvimento de software de qualidade. Modelos são construídos para comunicar a estrutura desejada, o comportamento do sistema, para visualizar e controlar a arquitetura e permitir entendimento do sistema

em construção, explorando oportunidades de simplificação, reuso e administrando riscos (BOOCH et al., 1999).

Aprender idéias básicas de modelagem não é difícil, mas não se pode dizer o mesmo quanto ao aprendizado de como usar e aplicar estas técnicas. A modelagem é particularmente complexa porque o modelador deve se aprofundar na natureza do negócio que está modelando. (HAY, 1996). JACOBSON et al. (1999) afirma que sistemas baseados em software são de difícil modelagem por não existirem em nosso mundo tridimensional. Freqüentemente tratam com questões únicas e sem precedentes, utilizam um conjunto de tecnologias frágeis e tem que possibilitar mudanças rápidas. Sistemas cada vez maiores e mais complexos necessitam ser modelados para criar uma visão comum do sistema entre os desenvolvedores, possibilitando o entendimento, organização, promovendo a reutilização e sendo capaz de permitir melhorias (JACOBSON et al., 1999).

Sistemas estão se tornando cada vez mais complexos e os problemas da modelagem estão indo além dos algoritmos e das estruturas de dados da computação: a especificação e o projeto das estruturas de sistemas estão emergindo como um novo tipo de problema (GARLAN; SHOW, apud JACOBSON et al., 1999). O coração e a alma dos efetivos modelos orientados a objetos são suas estratégias e seus padrões, não os formatos dos ícones ou o número de adornos que o modelo contém (COAD et al., 1995).

BOOCH et al. (1999) afirmam que uma empresa de software de sucesso é aquela que consistentemente desenvolve sistemas que atendam às necessidades dos usuários. Diagramas de fácil entendimento, reuniões estruturadas, programação com técnicas apuradas e todo o resto, é secundário, mas não irrelevante. Para desenvolver um sistema que atenda às necessidades do usuário necessita-se que o mesmo se engaje no projeto de maneira efetiva e organizada, definindo os reais requisitos do sistema. O desenvolvimento de um software com qualidade requer uma estrutura que resista a mudanças. Desenvolver rapidamente, eficientemente e efetivamente, com o mínimo de re-trabalho e perdas, é necessário ter as pessoas certas, ferramentas certas e o foco correto. Para fazer tudo isto de forma consistente e previsível necessita-se de um

processo de desenvolvimento de software que se adapte às mudanças dos negócios e das tecnologias (BOOCH et al.,1999).

2.9 Conclusão

Conforme FOWLER e HIGHSMITH (2001) facilitar as mudanças é mais eficaz que tentar impedi-las. Aprender a confiar na habilidade de responder a eventos imprevisíveis é mais importante que confiar na habilidade de se planejar um desastre.

Num ambiente mutável como a informática, onde avanços tecnológicos estão freqüentemente sendo introduzidos, parece estranho falar sobre soluções passadas e não sobre as novas tecnologias que surgirão (FOOTE, 1998). Enquanto nas universidades é comum o uso de novas tecnologias, empresas estão mais voltadas a utilizar padrões testados e aprovados (FOOTE, 1998). Um dos elementos chave dos padrões é que são criados a partir da observação do dia a dia do desenvolvimento, ao invés de uma invenção ou inovação acadêmica (FOWLER, 1997).

O uso de padrões, como pode ser visto, tem papel fundamental para a construção de sistemas cada vez mais personalizados, flexíveis, com baixo custo, com qualidade e velocidade na entrega. Se na indústria automobilística os padrões são um dos alicerces da sua evolução, na informática os mesmos também possuem papel fundamental para a sua evolução, por mais contraditório que possa parecer.

No próximo capítulo a UML será descrita, a qual será a linguagem de modelagem utilizada neste trabalho. O motivo da opção pela UML como linguagem de modelagem foi o fato da mesma ter se tornado um padrão de fato, facilitando o entendimento de maneira geral dos modelos.

3 Linguagem de Modelagem Universal (Unified Modeling Language - UML)

Como vimos no capítulo anterior, a modelagem tem papel fundamental no desenvolvimento e comunicação dos padrões, assim sendo, neste capítulo a UML será rapidamente descrita, para posterior utilização da mesma como linguagem de modelagem deste trabalho.

3.1 Histórico da UML

BOOCH et al. (1999) relatam que os métodos de modelagem orientados a objetos começaram a aparecer entre meados da década de 70 e final da década de 80, dada a necessidade de adequação dos mesmos a nova geração de linguagens orientadas a objetos e ao aumento da complexidade dos sistemas. No período entre 1989 e 1994, o número de métodos relatados saltou de menos de 10 para mais de 50, criando a chamada "Guerra dos Métodos" (BOOCH et al., 1999). Apesar desta diversidade, muitos usuários tinham dificuldades para encontrar uma metodologia que os atendesse por completo. Desta necessidade, e pela experiência adquirida, uma nova geração de métodos começou a surgir, destacando-se os de Booch, Jacobson (Object-Oriented Software Engineering - OOSE), Rumbaugh (Object Modeling Technique - OMT), Shlaer-Mellor, Coad-Yourdon e Fusion. Estes eram métodos completos e que possuíam pontos fortes e limitações. O método de Booch destacava-se na fase de projeto e de desenvolvimento, enquanto o OOSE destacava-se pela modelagem de alto nível e captura dos requisitos dos sistemas utilizando-se dos casos de uso, o OMT era mais voltado à análise de sistemas com uso intensivo de dados (BOOCH et al., 1999).

Conforme FOWLER e SCOTT (2000), em 1994 começou-se a falar na padronização dos métodos. Uma tentativa por parte da Object Management Group (OMG) causou até protestos por parte dos metodologistas. Booch tentou uma aproximação informal dos metodologistas, mas também não obteve sucesso. Neste mesmo ano, na OOPSLA'94, a grande notícia era que Rumbaugh estava deixando a General Eletric (GE) e se unindo a Booch na Rational Software, com a intenção de reunirem seus métodos. Como os métodos de Booch e o OMT de Rumbaugh estavam

crescendo e sendo reconhecidos pela comunidade usuária como métodos de classe mundial, ao se unirem, tornaram-se uma grande força que resultou em 1995 na versão 0.8 da Unified Method Documentation. Neste ano juntou-se a eles também Ivar Jacobson. Desde então, os três amigos, como passaram a ser conhecidos, trabalharam no novo método, chamado UML. Em junho de 1996, foi lançada a versão 0.9 da UML. Este esforço para a padronização ganhou parceiros importantes incluindo, Microsoft, IBM, Oracle, Hewlett-Packard, Texas Instruments, Sternilg Software, Unisys entre outros. Essa colaboração produziu, em janeiro de 1997, a UML 1.0 e a UML 1.1. Em setembro do mesmo ano a mesma foi aprovada pela OMG como padrão oficial. Desde então foi criada uma força tarefa, liderada por Cris Kobryn, que fez algumas revisões incrementais e que resultou na UML 1.2 e 1.3, sendo esta última a versão atual da UML, publicada em 1999 (FOWLER; SCOTT, 2000).

Quando Grandy Booch, James Rumbaugh, and Ivar Jacobson desenvolveram a UML, eles esperavam criar um padrão de modelagem que reunisse as melhores práticas da indústria e também ajudasse a desmistificar o processo de modelagem de software (FOWLER; SCOTT, 2000). A UML recebeu influência da modelagem de dados (diagrama de Entidade Relacionamento), de negócios (diagrama de Workflow) e de objetos e componentes, incorporando a idéia de vários autores (FURLAN, 1998). FURLAN (1998) afirma que "...os fomentadores da UML não inventaram a maioria das idéias, em vez disso, seu papel foi de selecionar e integrar as melhores práticas do mercado".

KOBRYN (1999) afirma que, num curto espaço de tempo, a UML se tornou a linguagem de modelagem dominante na indústria de software, se tornando um padrão não só pelo uso, mas também pelos órgãos de padronização. Além de ter sido adotada pela OMG como linguagem padrão, foi submetida à aprovação pela International Organization for Standardization (ISO) para, desta forma, tornar-se formalmente reconhecida como padrão internacional para a tecnologia de informação. Este reconhecimento traz como principal benefício a ampla aceitação da UML globalmente (KOBRYN, 1999).

3.2 Definição de UML

> "A UML é uma linguagem gráfica para a visualização, especificação, construção e documentação dos artefatos de sistemas com uso intensivo de software. A UML fornece uma forma padrão de documentar o sistema, cobrindo a parte conceitual, os processos de negócios, as funções do sistema e também a parte mais concreta: como escrever as classes em uma linguagem específica, esquema de banco de dados e componentes reutilizáveis de software." (BOOCH et al., 1999)

Na UML, conceitos são representados como símbolos, e relacionamentos entre os mesmos são representados como caminhos (linhas) que conectam os símbolos (ALHIR, 1999). Os vários diagramas são desenhados para representar um sistema sob diferentes perspectivas, representando uma visão resumida dos seus elementos. A UML é composta por nove tipos diferentes de diagramas que são os que seguem:

a) Diagrama de Classe - descreve os tipos de objetos do sistema e os vários tipos de relacionamento entre eles (FOWLER; SCOTT, 2000). Este é o diagrama mais comum na modelagem de sistemas orientados a objetos e se destina à modelagem da visão estática do sistema (BOOCH et al., 1999).
b) Diagrama de Objetos - representa um retrato dos objetos de um sistema em um determinado momento no tempo. Normalmente é utilizado para exemplificar a configuração dos objetos de uma classe mais complexa (FOWLER; SCOTT, 2000).
c) Diagrama de caso de uso - descreve a funcionalidade do sistema percebida pelos atores externos (usuários, dispositivos, demais sistemas) e os relacionamentos entre os mesmos e os casos de uso (FURLAN, 1998).
d) Diagrama de seqüência e de colaboração - são diagramas de interação, os quais mostram a interação constituída por um conjunto de objetos e seus relacionamentos, incluindo as mensagens que podem ser disparadas entre eles. Estes diagramas são destinados a modelar a visão dinâmica do sistema (BOOCH et al., 1999).
e) Diagrama de estado - mostra as seqüências de estados que um objeto ou uma interação assume em sua vida, em resposta a estímulo recebido, juntamente com suas respostas e ações (FURLAN, 1998). É especialmente útil para modelar sistemas reativos.

f) Diagrama de atividades - são usados para explorar e descrever o fluxo de processos de negócios dentro do contexto da organização (ERIKSSON; PENKER, 2000).
g) Diagrama de componentes - mostra a organização e dependência entre um conjunto de componentes (BOOCH et al., 1999). Componentes na UML são elementos físicos como um código fonte, bibliotecas, programas executáveis e componentes dinâmicos (ERIKSSON; PENKER, 2000).
h) Diagrama de implementação - mostra uma visão dos equipamentos num sistema (ERIKSSON; PENKER, 2000).

3.3 Diagramas da UML

3.3.1 Diagrama de Caso de Uso

O diagrama de caso de uso mostra a visão externa do sistema, de forma não orientada a objetos e sem compromisso com a organização das classes do sistema (FOWLER; SCOTT, 2000). Com isso, o sistema é representado como se fosse uma caixa preta, pode-se visualizar os elementos externos e como o mesmo reage aos eventos, não detalhando seu funcionamento interno. O diagrama de caso de uso é normalmente utilizado para modelar o contexto no qual o sistema está inserido e especificar seus requisitos (BOOCH et al., 1999).

REED (2000) coloca que o diagrama de caso de uso focaliza **o quê** o sistema deverá fazer e não **como** ele o fará. Os casos de uso são o centro a partir do qual são derivados todos os requisitos do sistema, sendo os mesmos tecnologicamente independentes e orientados a objetivos (REED, 2000).

Um dos principais objetivos dos diagramas de caso de uso é facilitar a comunicação entre os usuários e os desenvolvedores do sistema, modelando apenas os aspectos essenciais do mesmo, não sendo apropriado para modelar o fluxo de trabalho do sistema (MARSHALL, 2000).

Os casos de uso são documentos que narram e descrevem a seqüência de eventos de um ator que usa o sistema para completar um processo (JACOBSON et al., 1992). A UML não especifica um formato rígido para a documentação dos casos de

uso, os mesmos normalmente são representados por um texto, no qual a sua forma pode ser alterada conforme as necessidades da documentação e clareza na comunicação (LARMAN, 1998). Os casos de uso devem descrever o fluxo dos eventos, em texto claro o suficiente para que os usuários do sistema tenham um fácil entendimento do mesmo (BOOCH et al., 1999).

Os caso de uso são essenciais para a captura dos requisitos, planejamento e controle do projeto, sendo uma das primeiras ferramentas a serem utilizadas no desenvolvimento de sistemas (FOWLER; SCOTT, 2000). Salienta-se que apesar da importância dos diagramas de caso de uso, que servem para dar uma visão do contexto (BOOCH et al., 1999), os mesmos não dizem muito e não são documentos interessantes se não forem acompanhados das devidas descrições textuais (MARTIN, 1998b).

3.3.2 Diagrama de Atividades

BOOCH et al. (1999) afirmam que nenhum sistema com uso intensivo de software existe isoladamente, sempre há um contexto no qual o mesmo está inserido, especialmente os sistemas corporativos, os quais se inserem num contexto de alto nível nos processos de negócios das empresas. Os processos de negócios são representados pelos fluxos de trabalho e de objetos que fluem pelo negócio, podendo ser modelados utilizando-se dos diagramas de atividades que servem para visualizar, especificar, construir e documentar os mesmos (BOOCH et al., 1999).

Os diagramas de atividades também são utilizados para modelar os algoritmos das operações, facilitando o entendimento do código a ser desenvolvido. É através do diagrama de atividades que é possível gerar o código automaticamente e também fazer engenharia reversa do código já existente (BOOCH et al., 1999).

Conforme AMBLER (2000b) os diagramas de atividades servem para documentar uma operação ou método, um caso de uso, ou o fluxo de trabalho de um processo de negócios.

FOWLER e SCOTT (2000) aconselham utilizar o diagrama de atividades antes de modelar os casos de uso, aplicando o mesmo em conjunto com os

especialistas do negócio para entender o processo no qual o sistema está envolvido e avaliar as possíveis mudanças no processo. Além disso, os autores aconselham o uso do diagrama de atividades para analisar os casos de uso, descrever algoritmos complexos e modelar aplicações com paralelismo.

O diagrama de atividades é uma das áreas menos entendidas da UML (FOWLER; SCOTT, 2000), atualmente o mesmo é uma especialização do diagrama de estados (MARSHALL, 2000). Está situação está se alterando, pois na UML 2.0 o diagrama de atividade será semanticamente independente do diagrama de estados, fato este que facilitará a expansão de ambos os diagramas direcionando-os mais às suas finalidades específicas (KOBRIN, 1999).

3.3.3 Diagrama de Estados

O diagrama de estados é o diagrama que tem sido utilizado por mais tempo, sua origem se deu da necessidade dos projetistas de circuitos integrados complexos modelarem os estados em que o sistema poderia estar em qualquer instante específico como também os eventos aos quais o sistema deveria responder, quando em um determinado estado, e o trabalho a ser executado na mudança de estado. Seu uso inicial não possuía associação nenhuma com os conceitos orientados a objeto (REED, 2000).

O diagrama de estados pode ser utilizado para representar um sistema, um caso de uso ou um objeto (BOOCH et al., 1999), caracterizando-se por armazenar um estado interno que representa as suas experiências passadas, funcionando como um sistema autômato no qual o comportamento de sua saída não é resultado apenas da entrada atual, mas também conseqüência do histórico das entradas anteriores (OVERGAARD et al., 2000).

Adaptados aos conceitos orientados a objetos, os diagramas de estado, descrevem todos os estados possíveis em que um objeto em particular pode estar e como os estados deste objeto se alteram em decorrência de eventos que chegam ao mesmo. Na orientação a objetos os diagramas de estado modelam o comportamento de um objeto individual em seu ciclo de vida (FOWLER; SCOTT, 2000).

Nem todas as classes são adequadas para serem representadas em um diagrama de estados, uma classe tipo CLIENTE talvez possa estar em um estado ativo ou inativo, não sendo tão elucidativo um modelo dos estados desta classe, já uma classe tipo PEDIDO pode estar em vários estados diferentes em seu ciclo de vida (pedido não confirmado, confirmado, cancelado, atendido...) e será influenciada por vários tipos de eventos diferentes no sistema, sendo desta forma aconselhável a utilização de um diagrama de estados (REED, 2000).

A UML possui uma notação muito rica para descrever os diagramas de estado, a mesma inclui elementos de fluxogramas e de redes de Petri, é a mais completa notação reunida em uma única notação, possibilitando uma representação muito densa. Muito da lógica de um sistema pode ser representado em um simples diagrama (MARTIN, 1998a).

Geralmente sistemas de informações empresariais possuem poucos objetos dependentes de estado interessantes, ao contrário dos sistemas de controle de processos e de telecomunicações que possuem um grande número (LARMAN, 1998).

3.3.4 Diagramas de Interação

Caso seja necessário analisar o comportamento de um único objeto através de vários casos de uso, utiliza-se o diagrama de estados; se desejar analisar o comportamento do sistema através dos casos de uso, utiliza-se o diagrama de atividades e se desejar analisar o comportamento de vários objetos em um caso de uso utilizam-se os diagramas de interação (FOWLER; SCOTT, 2000).

Os diagramas de interação mostram as interações entre um conjunto de objetos, incluindo as mensagens disparadas entre os mesmos (BOOCH et al., 1999). Os objetos podem ser unidades organizacionais, empresas, computadores, pessoas, processos ou elementos de software (ERIKSSON; PENKER, 2000). Podem ser utilizados para modelar os cenários dos casos de uso (BOOCH et al., 1999) assim como os processos empresariais em trabalhos de reengenharia (REED, 2000).

Existem dois tipos de diagramas de interação: o diagrama de seqüência e o diagrama de colaboração (FOWLER; SCOTT, 2000). O diagrama de seqüência

enfatiza a ordem das mensagens no tempo, focalizando o fluxo linear de mensagens entre os objetos, sendo que o tempo decorre do início do diagrama seqüencialmente para baixo (REED, 2000), já o diagrama de colaboração enfatiza a organização estrutural dos objetos que enviam e recebem as mensagens. Os dois tipos de diagramas são semanticamente equivalentes, podendo ser convertidos de um para o outro sem perda de informações (BOOCH et al., 1999).

3.3.5 Diagrama de Classe

Os diagramas de caso de uso, atividades, seqüência, colaboração e estados (vistos anteriormente) são os diagramas responsáveis por modelar os aspectos dinâmicos do sistema, já os diagramas de classe, componentes, implantação e de objetos são utilizados para modelar a visão estática do sistema. O diagrama de classe é o diagrama mais comumente utilizado para modelar sistemas orientados a objetos e também serve de base para os diagramas físicos de componentes e implantação, já o diagrama de objetos é considerado uma instância do diagrama de classes (BOOCH et al., 1999).

O diagrama de classe é utilizado sob perspectivas diferentes na fase de análise, projeto e implementação do sistema. Na fase de análise, o mesmo é utilizado para construir o modelo conceitual do sistema, o qual é o mais importante artefato orientado a objetos a ser criado durante a fase de análise. O modelo conceitual é composto pelas classes, associações entre classes e os atributos das classes, se concentrando em modelar o mundo real, que faz parte do domínio do problema, e não o projeto do sistema com as classes específicas das linguagens, como Java ou C++, ou ainda as telas ou o banco de dados (LARMAN, 1998).

Para construir um diagrama de classe bem estruturado devem-se seguir algumas convenções de modelagem. Apesar destas não fazerem parte das regras da UML, é aconselhável segui-las para ter-se uma melhor comunicação e tornar o trabalho do modelador mais simples (HAY, 1996).

3.3.6 Diagrama de Objetos

O objetivo do diagrama de objetos é explicar e exemplificar diagramas de classes complexos. O diagrama de objetos é um retrato dos objetos de um sistema e de seus relacionamentos em um determinado ponto no tempo. O mesmo é desenhado com objetos e ligações entre eles, sendo as ligações, instâncias das associações e agregações. Como o diagrama de objetos é um retrato, a multiplicidade dos relacionamentos não é mostrada, ao invés disto são mostradas as ligações entre os objetos que compõem o retrato do sistema naquele instante.

O diagrama de objetos é essencialmente uma instância do diagrama de classe (BOOCH et al., 1999).

3.3.7 Diagrama de Componentes

Quando se modela um sistema, deve-se considerar tanto a dimensão lógica quanto a física. A dimensão lógica é composta por elementos como classes, interfaces, interações, colaborações e máquinas de estado. Já a dimensão física é composta por componentes que representam o agrupamento físico dos elementos lógicos e pelos nós que representam o hardware onde os componentes serão executados.

Os componentes são elementos físicos, substituíveis, que fazem parte de um sistema residindo em seus nós (nodes). Alguns exemplos de componentes são: componentes COM+ ou Java Beans, programas executáveis, bibliotecas, tabelas, arquivos e documentos.

O diagrama de componentes é um diagrama de classes com todos os seus elementos, sendo que os componentes se diferenciam das classes por:

a) Classes representam abstrações lógicas, já os componentes representam elementos físicos, que fazem parte do mundo dos bits, residindo em nós do sistema.
b) Os componentes reúnem outros elementos lógicos (como classes) estando num nível de abstração diferente.

c) Classes podem ter atributos e operações diretamente, já os componentes, em geral, possuem apenas operações que podem ser acessadas através da sua interface.

Se o sistema que está sendo desenvolvido consiste apenas de um programa executável, não existe a necessidade de se desenhar o diagrama de componentes. O mesmo deve ser utilizado quando se possui vários programas executáveis, bibliotecas e componentes interagindo entre si. Neste cenário, o diagrama de componentes servirá para modelar as decisões feitas em relação ao mundo físico.

3.3.8 Diagrama de Implantação

O diagrama de implantação utiliza nós para modelar a topologia do hardware no qual o sistema é executado. Os nós tipicamente representam computadores ou dispositivos do sistema, que geralmente possuem memória e capacidade de processamento, onde podem ser disponibilizados os componentes.

O diagrama de implantação é um diagrama de componentes, com todos os seus elementos, sendo que os nós se diferenciam dos componentes por:

a) os componentes são elementos que participam da execução do sistema, os nós são os elementos que executam os componentes;
b) os componentes representam o agrupamento físico dos elementos lógicos, já os nós disponibilizam fisicamente os componentes.

O tipo mais comum de relacionamento entre os nós é a associação, que neste contexto representa a conexão física entre os nós, podendo a mesma ser do tipo Ethernet ou linha serial, entre outras. Os nós, como as classes, podem ter atributos e operações, que neste contexto representam atributos como “tipo do processador”, “memória” ou operações como “desligar” ou “suspender”.

Normalmente, o diagrama de implantação é desenhado utilizando-se de vários tipos de ícones, que se parecem com os elementos físicos que estão sendo modelados, como PCs, banco de dados e servidores, entre outros. Esta forma de representação é permitida na UML através do uso de estereótipos, que podem ser definidos pelo

modelador, tornando o diagrama mais simples de ser entendido (FOWLER; SCOTT, 2000)

Os diagramas físicos podem ser desenhados isoladamente ou em conjunto, desta forma mostram quais componentes rodam em cada computador (FOWLER; SCOTT, 2000). A maioria das pessoas desenha informalmente os diagramas físicos, mas gradualmente a UML está sendo adotada para tal finalidade (FOWLER; SCOTT, 2000).

3.4 Prática de Uso na Utilização da UML na Definição de Padrões

Conforme BOOCH (1999), a classe é o mais importante elemento de construção de qualquer sistema orientado a objetos. Ela é a descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica, é uma abstração dos elementos que fazem parte do vocabulário, não é um objeto em particular mas sim, representa todo um conjunto de objetos (BOOCH, 1999). Na modelagem de padrões o diagrama de classe é o mais utilizado sendo apoiado pelo diagrama de objetos e de interação (ERIKSSON; PENKER, 2000).

Como, atualmente, a maioria dos sistemas utiliza banco de dados relacionais para armazenar os dados de forma persistente, todos os diagramas de classe conterão sua conversão para um "Diagrama lógico persistente" que demonstra como o sistema foi modelado para uso com um banco de dados relacional. Os "Diagramas lógicos persistentes" foram modelados em UML, com o uso de esteriótipos baseados nos artigos da RATIONAL (2000) e AMBLER (1999). No decorrer do texto, considerações quanto à utilização do banco de dados relacional também serão feitas.

3.5 Conclusão

ALHIR (1999) afirma que o mercado globalizado se caracteriza pelas rápidas mudanças e aumento da complexidade dos negócios. Neste contexto, o conhecimento se tornou fator fundamental. Organizações não podem mais ser baseadas em músculos, mas tem que saber aproveitar o capital intelectual criativo para obter um diferencial competitivo. O valor do conhecimento é demonstrado através da sua aplicação na

solução de problemas. Se o conhecimento é adquirido e aplicado pela organização, esta se torna mais competitiva e proativa.

O uso da UML facilita a comunicação entre profissionais de várias áreas, tais como administradores, programadores, analistas de sistemas, assessores e outros com diferentes características. Apesar da UML possuir uma notação rica, que objetiva a definição de sistemas, de maneira geral, esta pode ser utilizada de forma intuitiva por profissionais e servir como base da especificação de software (HRUBY, 2000).

O sucesso da UML será medido pelo uso apropriado em sistemas bem sucedidos. A adoção da UML para a modelagem de sistemas não é garantia de sucesso, mas habilita seus usuários a construírem diagramas úteis, utilizando uma linguagem de modelagem consistente, padronizada e suportada por ferramentas de desenvolvimento (ALHIR, 1999).

4 Exercício com Padrões de Análise

Atualmente existem diferentes tipos de padrões, cada um com propósito diferente, sendo utilizado em fases distintas do processo de desenvolvimento de software. ERIKSON e PENKER (2000) classificam padrões em três categorias:

a) Padrões de análise: direcionados à solução de problemas de negócios, tipicamente situações de análise de sistemas, sendo similares à modelagem de estruturas empresariais.
b) Padrões de arquitetura: direcionados para a área de projeto do sistema de informações, sendo utilizados para orientar a organização dos subsistemas ou mesmo definir a estrutura de implementação em alto nível de abstração.
c) Padrões de projeto: usados em situações onde a análise foi concluída e se busca a produção de soluções técnicas flexíveis e adaptáveis.

Os modelos que serão apresentados neste trabalho são o resultado da utilização prática de padrões de análise na composição de um sistema real. Os modelos são apresentados com a respectiva fundamentação teórica e sua utilização prática. Os padrões fundamentam os modelos propostos os quais serão descritos de forma evolutiva para facilitar o entendimento.

Este capítulo começa descrevendo os modelos básicos que vão se relacionando, formando no final um modelo único. Os modelos apresentados neste trabalho se dividem em dois grupos, os modelos de apoio e os modelos principais. Os modelos de apoio se caracterizam por criar uma estrutura de tipos que servirão de apoio aos modelos principais. Os modelos de apoio são os que seguem:

Modelo Empresas e Usuários, serve para controlar os direitos de acessos de cada usuário aos dados do sistema, o mesmo se relaciona com todos os demais modelos através de classes específicas a cada modelo.

Modelo Histórico (usualmente conhecido como *log),* se destina a manter o histórico das modificações sobre os dados do sistema e a armazenar qualquer evento desejado. O mesmo define quatro tipos de identificadores, sendo obrigatório o uso de um deles pelas classes que se deseja manter histórico.

Modelo Unidades, define o tipo **QUANTIDADE** que é utilizado em todos os atributos numéricos do sistema. O mesmo armazena o valor desejado como também sua unidade de medida, podendo ser esta física ou financeira.

Os modelos principais são responsáveis pela modelagem do sistema propriamente dito, sendo os mesmos os que seguem:

Modelo Recursos, define de forma hierárquica todos os tipos de recursos, sendo estes físicos ou informacionais, que podem ser transacionados pelo sistema. Exemplos de recursos são: produtos, fornecedores, clientes, contas contábeis e centros de custos.

Modelo Documento, armazena os dados e relacionamentos de todos os documentos ou transações do negócio, tendo sua origem no modelo contábil. Exemplos de documentos são: notas fiscais, apontamentos de produção, pedidos, requisições e exames.

Modelo Caráter, caracteriza um recurso ou documento e é responsável por armazenar as propriedades dos mesmos. Como exemplo um recurso do tipo Pessoa pode ser caracterizado através de seu peso, altura, endereço, data de nascimento entre outros.

Estes modelos podem inicialmente parecerem de difícil entendimento, mas são estruturas simples e flexíveis, que se adaptam as mais diversas formas de negócios. Nos tópicos que se seguem será detalhado o uso destes.

4.1 Controle de Acesso

Com a globalização e o advento do comércio eletrônico cada vez mais empresas estão expandindo-se em novos negócios, abrindo novas unidades e disponibilizando seus sistemas a um número de usuários crescente através da Internet. Estes fatos geram a necessidade de sistemas distribuídos, que sejam capazes de suportar mais de uma empresa no mesmo modelo e possam controlar o acesso aos seus dados de forma apropriada. Os modelos empresas e usuários que serão vistos a seguir servem para administrar estas complexidades.

4.1.1 Modelo Empresas

Em sistemas que suportam mais de uma empresa, de forma distribuída ou não, uma das decisões a ser tomada é quanto ao compartilhamento dos objetos do sistema. Se a empresa está abrindo uma nova filial, com a mesma atividade da matriz, provavelmente os cadastros serão compartilhados. Se a empresa está abrindo um novo negócio, completamente distinto do atual, pode não ter sentido compartilhar cadastros. Como esta sendo criado um sistema flexível capaz de suportar alterações no negócio, é desejável que o mesmo se adapte a diferentes formas de compartilhamento de dados entre empresas diferentes e usuários com necessidades distintas.

Uma empresa que está em crescimento e resolve criar uma nova unidade de negócio independente de sua matriz, pode simplesmente criar um novo banco de dados ou esquema tornando assim os dados desta independentes das demais empresas do grupo.

Apesar desta solução resolver o problema nem sempre é possível criar novos esquemas ou banco de dados para cada novo negócio. Em um escritório de contabilidade, que gerencia várias empresas simultaneamente, é necessário ter uma estrutura, que torne os dados independentes entres as empresas que compartilham o mesmo esquema, sem a necessidade de criação de uma nova base de dados ou novo esquema.

Para que se possa distinguir se empresas compartilham os mesmos dados ou não tem-se a classe EMPRESA (FIGURA 1), que está associada a quase todas as classes do sistema, fazendo com que ao se criar uma nova empresa esta possuirá uma independência completa das demais.

No modelo relacional as tabelas terão em sua chave primária a chave estrangeira associada à tabela EMPRESA. Esta estrutura faz com que todos os objetos associados à EMPRESA ‘01-fábrica’, por exemplo, sejam independentes da EMPRESA ‘02-distribuidora’ e das demais EMPRESAS.

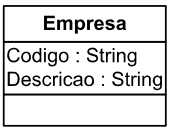

FIGURA 1 - Classe Empresa

Quando uma unidade de negócio nova, necessitar utilizar os dados de uma unidade de negócio já existente, será criado uma nova divisão ou sub-DIVISÃO associada a mesma EMPRESA da divisão já existente. Como podemos ver na FIGURA 2, toda a DIVISÃO está associada a uma EMPRESA , as DIVISÕES podem possuir sub-divisões, determinadas pelo relacionamento recursivo *pai*. As DIVISÕES associadas a uma mesma EMPRESA podem ou não compartilharem os seus dados, dependendo da classe em questão e dos cadastros associados a mesma. A utilização da classe DIVISÃO será detalhada após cada modelo no tópico que trata sobre o controle de acesso a classe.

Para simplificar, se não houver compartilhamento de dados entre as DIVISÕES as mesmas deverão estar ligadas a EMRESAS distintas, se houver a necessidade de compartilhamento de dados as DIVISÕES deverão estar ligadas a mesma EMPRESA sendo que o compartilhamento ou não dos dados seguirá regras especiais associadas a cada modelo em específico.

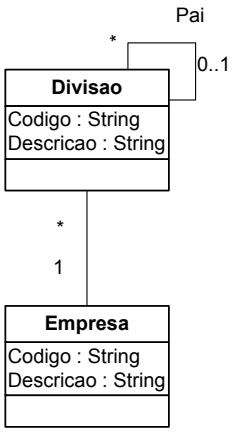

FIGURA 2 - Modelo Empresa com Divisão

4.1.2 Modelo Usuários

Apesar dos bancos de dados atuais também contemplarem o gerenciamento do acesso dos usuários, optou-se em criar a classe USUÁRIO de forma independente da

estrutura do banco de dados. Isto foi feito para facilitar o gerenciamento por parte da aplicação e melhor controlar o acesso do usuário tanto aos objetos do banco de dados como também aos objetos da aplicação (como será visto posteriormente no item Controle de Acesso).

Todos os usuários do sistema deverão estar associados a uma EMPRESA, e serão identificados pela EMPRESA a que estão associados e seu **código**. Um usuário não necessita estar ligado a uma DIVISÃO específica, o que lhe da direito a acessar qualquer DIVISÃO associada a EMPRESA que o mesmo está associado, já se o mesmo estiver ligado a uma DIVISÃO específica somente acessará os dados desta e de suas sub-DIVISÕES.

Como podemos ver na FIGURA 3, uma EMPRESA possui vários USUÁRIOS que são de um determinado TIPO USUÁRIO. À classe TIPO USUÁRIO estarão associados todos os direitos de acesso permitidos ao USUÁRIO (a estrutura dos direitos de acesso são particulares a cada modelo), sendo que o mesmo poderá herdar também os direitos de acesso de seu *pai*, dado através da estrutura hierárquica da classe USUÁRIO. Assim sendo, um USUÁRIO terá direito de acesso a todos os objetos que seu TIPO USUÁRIO permite e herdará os direitos de acesso de seu USUÁRIO *pai*.

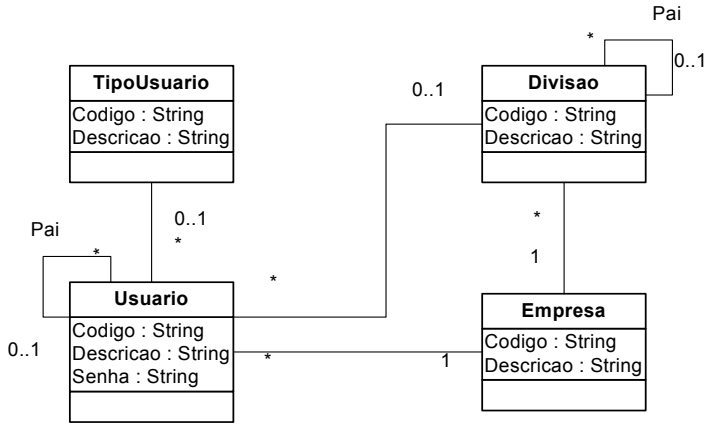

FIGURA 3 - Modelo Usuários

4.2 Histórico das transações

4.2.1 Padrão Histórico das transações

Como os dados em um sistema estão em mudança contínua, deve-se ter cuidado com os dados que se alteram no decorrer do tempo. Quando os dados mudam, existem circunstâncias nas quais é necessário manter histórico das alterações. As alterações do nome de uma pessoa ou do valor do seu salário são exemplos de dados que devem ser associadas a um histórico. Este é muito importante para a correção de erros e para visualizar a situação anterior de um objeto (DORSEY; HUDICKA, 1999).

Conforme DORSEY e HUDICKA (1999), existem seis formas de se manter um histórico em qualquer sistema, que são as que seguem:

a) **Fazer parte do próprio modelo** - o modelo, por sua natureza, possui uma estrutura que mantém o histórico das transações, não sendo necessária nenhuma alteração para criar o histórico.

b) **Adicionar atributos para manter o histórico** - em classes de associação já existentes, ao adicionar os atributos **DataInício** e **DataFim** cria-se o histórico das alterações da classe, possibilitando também que se faça lançamentos com datas futuras ou passadas.

c) **Adicionar classe de cruzamento** - A idéia é alterar um relacionamento de 1-para-vários, para vários-para-vários, ou seja, em um modelo com um relacionamento simples, transforma-se o mesmo em uma classe com o atributo **DataDeInclusão**. Esta classe é chamada de classe de cruzamento e conterá o histórico das alterações do relacionamento.

d) **Duplicar as classes** - Este método consiste em criar duplicatas exatas das classes das quais o histórico deverá ser mantido, adicionando às mesmas dois atributos: a **DataDeInício** e a **DataDeFim**. A classe primária e a do histórico possuem uma associação de zero-ou-um para várias, sendo que a chave da classe do histórico é a mesma da classe primária, adicionada da data de início. Antes de cada alteração, em qualquer objeto da classe primária, o objeto é duplicado e inserido na classe de histórico com a

DataDeInício apropriada, o objeto anterior do histórico tem sua **DataDeFim** atualizada.

e) **Armazenar os objetos e seu histórico na mesma classe** - Esta técnica cria uma cópia do objeto, quando da sua alteração, na própria classe, sendo que o novo objeto criado possuirá um relacionamento recursivo com o objeto alterado e, por motivos de performance, será marcado com um indicador como sendo o novo registro corrente.

f) **Manter um histórico das transações (*log*) ou registro de auditoria** - A idéia é que sempre que algo significante acontece é gravado um registro que indica o que aconteceu.

O histórico das transações **(*log*)** é a forma mais simples e também uma das mais efetivas para a manutenção do histórico das transações. Comparando-se a outras soluções, o histórico das transações **(*log*)** não aumenta a complexidade do modelo, facilitando a sua implementação. O mesmo pode ter várias formas físicas, podendo ser um arquivo texto no formato ASCII, com uma estrutura tabular, uma estrutura mais complexa como um arquivo XML ou ainda pode ser armazenado em tabelas do banco de dados com uma estrutura definida (FOWLER, 2000a).

Para que se mantenha um histórico das transações**(*log*)** pode-se criar uma classe HISTÓRICO*,* a qual será responsável por armazenar todas as informações das transações efetuadas em qualquer outra classe do sistema. A classe HISTÓRICO pode ser implementada de forma simples, com os seguintes atributos:

- **Seqüência***,* número seqüencial criado pelo sistema;
- **Data**, data e hora da transação;
- **Usuário***,* responsável pela mudança;
- **TipoOperação,** pode ser de inclusão, alteração ou exclusão;
- **Classe** a qual se refere a alteração;
- **Chave**, identificador único do objeto que está sendo alterado;
- **Texto***,* contém a descrição das alterações efetuadas no objeto.

Todas as classes que geram lançamentos no HISTÓRICO devem possuir métodos que deverão ser disparados automaticamente após as alterações e que serão

responsáveis pela atualização do HISTÓRICO. Como exemplo, se no objeto 'João' fosse alterado o salário de '$1.000,00' para '$1.200,00' e a comissão de '$0' para '$500', poder-se-ia gerar o texto com a seguinte descrição: "SALARIO: 1000, 1200; COMISSAO: 0, 500". Para não ser oneroso em termos de espaço utilizado e tempo de processamento, pode-se gravar no HISTÓRICO apenas as alterações e as exclusões, não gravando as inclusões (DORSEY; HUDICKA, 1999).

Um padrão utilizado para armazenar os registros históricos das transações é mostrado na FIGURA 4. Nesse, as classes e as propriedades do sistema como um todo estão representadas em classes distintas, sendo que as alterações efetuadas em qualquer propriedade do sistema são armazenadas em duas classes, e não mais em uma única. A classe EVENTO DO HISTÓRICO armazena todos os eventos acorridos com os objetos do sistema sendo que na classe DETALHE DO EVENTO são armazenadas as alterações efetuadas em cada propriedade do objeto. Este padrão é capaz de recriar ou desfazer as transações facilmente, se armazenar todas as transações efetuadas no sistema (DORSEY; HUDICKA, 1999).

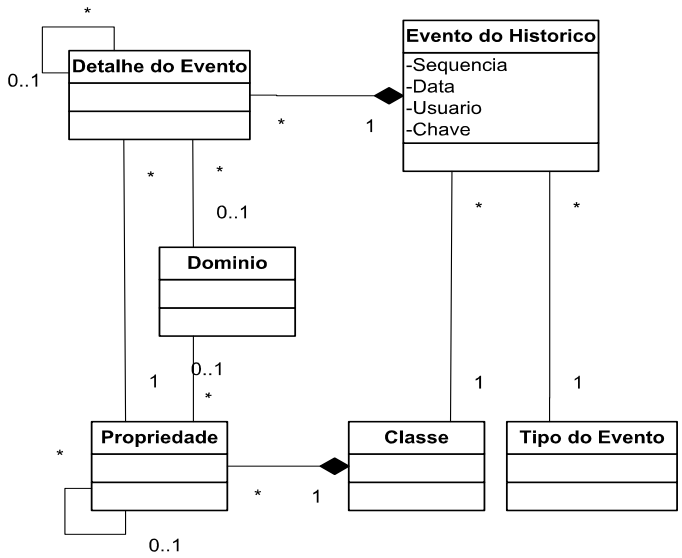

FIGURA 4 - Padrão Histórico das Transações (DORSEY; HUDICKA, 1999)

4.2.2 Tipos e Modelos para Armazenamento do Histórico

FOWLER (2000a) afirma que, quanto mais integradas estiverem as informações históricas ao modelo do sistema, menos útil será o histórico das

transações (*log*). Apesar disto, DORSEY e HUDICKA (1999) afirmam que, mesmo quando o modelo do sistema possue outras estruturas para manter o histórico, o histórico das transações (*log)* é altamente recomendado para todas as classes do sistema.

Apesar das afirmações parecerem opostas, optou-se por integrar ao modelo estruturas que são capazes de manter as informações históricas e conjuntamente ter um histórico das transações (*log*), o qual será responsável por armazenar apenas as informações históricas que não são armazenadas no próprio modelo.

O principal objetivo da utilização do histórico neste trabalho é que o mesmo possibilite a auditoria no sistema. Se forem armazenadas no histórico das transações (*log*), todas as transações de todas as classes, tem-se um histórico completo. A adoção deste procedimento no mínimo dobra o volume de dados no sistema, além de tornar a gravação das transações lenta. Como os sistemas de banco de dados atuais possuem sistemas próprios para a recuperação da base de dados, não se achou necessária a gravação de todas as transações no histórico das transações (*log*), optando-se por gravar no histórico das transações (*log*) apenas as alterações e as exclusões, política esta que não degrada a performance do sistema e que possibilita total auditoria das transações.

Para uma melhor integração das classes do modelo com o histórico das transações, optou-se por inserir em todas as classes o atributo **Id** que é de um dos tipos de identificadores que serão descritos a seguir. No modelo relacional esta política padroniza a chave primária de todas as tabelas facilitando a criação do histórico das transações e a convivência do modelo relacional com o modelo orientado a objetos.

4.2.2.1 Tipo Identificador

O tipo **IDENTIFICADOR** (FIGURA 5) é composto por um relacionamento com a classe USUÁRIO, que informa qual usuário efetuou a inclusão do objeto, o atributo **DataInclusão** que informa a data e hora da inclusão e o atributo **Código** que identifica o objeto, sendo que este é gerado automaticamente pelo sistema quando da sua inclusão. No modelo relacional, a coluna **Código** e **Empresa** compõem a chave

primária em todas as tabelas, sendo que a combinação de ambas será única em todo o sistema. Esta técnica é chamada de chave substituta (*surrogates keys*), técnica que facilita a conversão do modelo relacional para o de objetos (AMBLER, 1999), como também a gravação do histórico das transações.

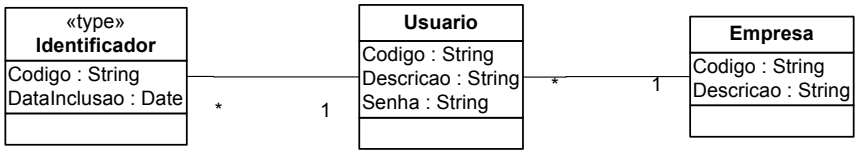

FIGURA 5 - Tipo Identificador

4.2.2.2 Tipo IDENTIFICADOR INDEPENDENTE

Para facilitar a modelagem, foi criado um tipo de dado padrão que é o **IDENTIFICADOR INDEPENDENTE**, conforme mostrado na FIGURA 6. Este tipo de dado é composto pelo atributo **Id** que é do tipo **IDENTIFICADOR**, e pelo atributo **Descrição**, que é do tipo **STRING**. O tipo **IDENTIFICADOR INDEPENDENTE** é utilizado em todas as classes que possuem uma identificação própria e independente das demais.

O **Id** do **IDENTIFICADOR INDEPENDENTE** possui o atributo **Código**, o qual é encontrado em todas as classes do sistema, mas possui um comportamento diferenciado nas classes que possuem atributo do tipo **IDENTIFICADOR INDEPENDENTE**. O **código**, nestas classes, opcionalmente é gerado pelo sistema, sendo permitido ao usuário informar o mesmo, desde que obedeça a restrição de não existir nenhum outro objeto no sistema com a **empresa** e **código** iguais. Optou-se por esta solução para que os **códigos** tenham sentido para o usuário quando o mesmo desejar , como exemplo de códigos significativos para o usuário pode-se citar as unidades de medida, 'R$', 'Us$', 'Kg' ou ainda uma codificação já existente na empresa que se deseja manter.

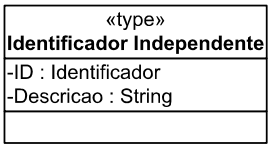

FIGURA 6 - Tipo Identificador Independente

4.2.2.3 Tipo IDENTIFICADOR HIERÁRQUICO

O **IDENTIFICADOR HIERARQUICO** é utilizado pelas classes independentes que necessitam uma organização hierárquica simples. Estas classes podem não fazer uso de sua estrutura hierárquica quando possuem poucos objetos, mas outras vezes o número de objetos pode ser significativo necessitando uma organização para que se possa administrar as mesmas. Esta estrutura também possibilita que os objetos filhos herdem propriedades de seus pais em situações específicas.

O **IDENTIFICADOR HIERARQUICO** é composto por um **IDENTIFICADOR INDEPENDENTE** e um auto-relacionamento (FIGURA 7). No modelo relacional é adicionada a coluna Pai que é responsável pela implementação do auto-relacionamento.

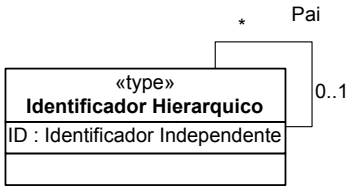

FIGURA 7 - Tipo Identificador Hierárquico

Salienta-se que a classe USUÁRIO, TIPO USUÁRIO e DIVISÃO utilizarão o **IDENTIFICADOR HIERÁRQUICO** como mostrado na FIGURA 8.

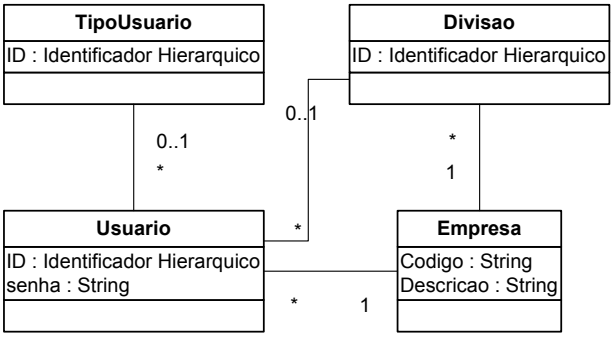

FIGURA 8 - Modelo Usuários com Identificador Hierárquico

Na FIGURA 9 podemos ver a conversão do Modelo Empresas e Usuários para o Modelo Empresas e Usuários Persistente (Banco de Dados Relacional).

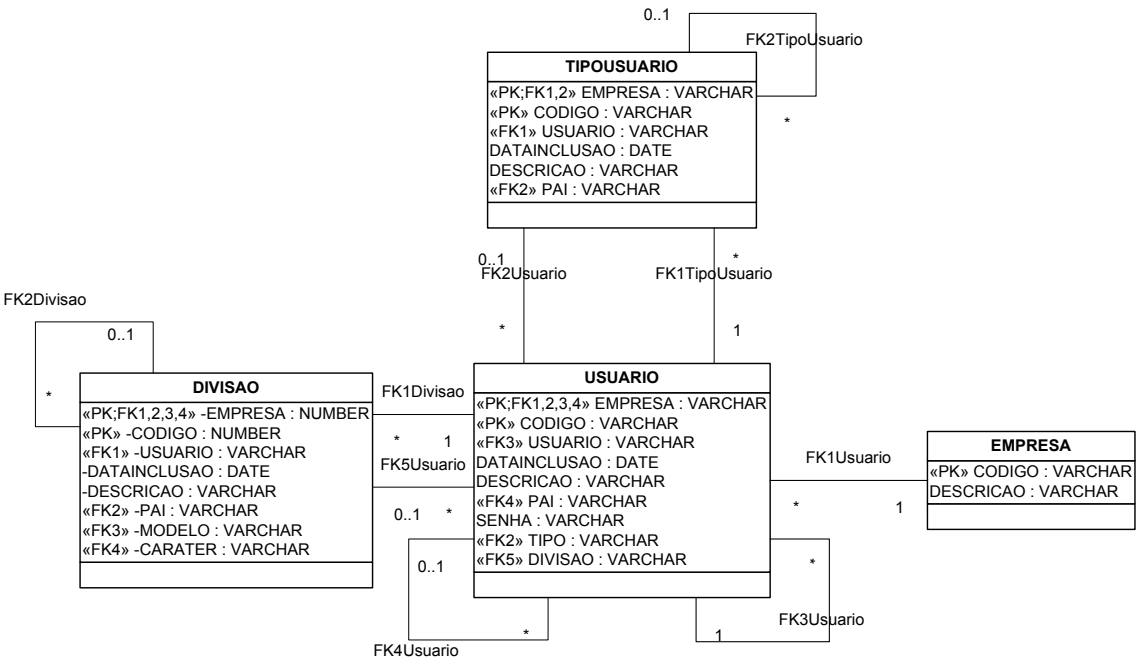

FIGURA 9 - Modelo Persistente Usuários (Banco de Dados relacional)

4.2.2.4 Tipo Identificador Temporal

Ao observar o mundo real pode-se notar que muitos dados cotidianos não são estáticos, ou seja, os mesmos variam com o passar do tempo, um dado que é válido hoje, pode ser inválido amanhã. Exemplificando, o peso de uma pessoa é válido no instante da sua medida, ao ser pesada, tempos depois, a medida provavelmente será diferente da original (FOWLER, 1997).

Como no mundo real, tem-se dados, que são verdadeiros somente durante um período de tempo, FOWLER (2000a) propôs que os mesmos fossem modelados juntamente com a data inicial e a final de sua validade. Esta é a forma mais comum de indicar a temporalidade em um modelo, como também a mais simples de ser utilizada e entendida. A principal desvantagem oferecida por este modelo é que ao ser consultada a base de dados, sempre deve-se indicar em que data se deseja o dado (FOWLER, 2000a).

Salienta-se que a data inicial e final da validade de um objeto não é a data da criação do objeto. Para que se modele tal fato necessita-se de dois registros de tempo, um para armazenar o tempo durante o qual o objeto é válido, utilizando-se do tipo

PERÍODO DE TEMPO (FIGURA 10) que possui os atributos **DataInicio** e **DataFim** e outro registro de tempo para armazenar quando foi criado o objeto, que pode ser representado por uma data simples FOWLER (1996).

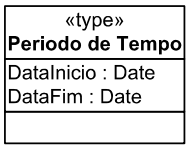

FIGURA 10 - Tipo Período de Tempo (FOWLER, 2000A)

Em todas as classes do sistema, que seus dados são válidos por um período de tempo, as mesmas possuem um atributo do tipo **IDENTIFICADOR TEMPORAL** (FIGURA 11). Este é composto pelo atributo **Período** do tipo **PERÍODO DE TEMPO (**FIGURA 10**)** em conjunto com o **Id** do tipo **IDENTIFICADOR (**FIGURA 5**)**. As classes que se utilizam do tipo **IDENTIFICADOR TEMPORAL** terão seu histórico mantido no próprio modelo, não sobrecarregando o Histórico das Transações.

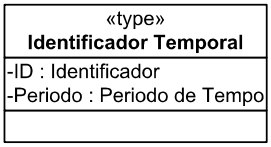

FIGURA 11 - Tipo Identificador Temporal

Por questões de performance e facilidade de implementação, é importante que se tenha a representação padrão para a data final em aberto ou data infinita (FOWLER, 2000a). Esta data, na verdade, indica que não há data de fim prevista, ou seja, na maioria das vezes quando se atribui um novo valor a uma dado não se sabe por quanto tempo ele será válido, nestes casos se padronizou que a **DataInicial** será igual a data corrente e a **DataFinal** será igual a '04/04/4444 00:00:00'. Quando este dado deixar de ser válido e outro for coletado, a **DataFinal** do registro anterior será igualada a **DataAtual** do novo registro que é igual a data corrente e a **DataFinal** terá seu valor igual a '04/04/4444 00:00:00'. Ressalta-se que esta é a forma mais comum de utilização do tipo **PERÍODO DE TEMPO**, mas o mesmo também é utilizado para que se efetuem lançamentos que valerão no futuro, que foram verdade no passado ou ainda com um prazo de validade pré-determinado. A utilização da data '04/04/4444

00:00:00’ como data infinita se deu por ser de fácil memorização, por estar distante da data atual e ser suportada pelos bancos de dados relacionais.

Na FIGURA 12 pode-se visualizar, em um modelo único, os relacionamentos existentes entre os vários tipos de identificadores.

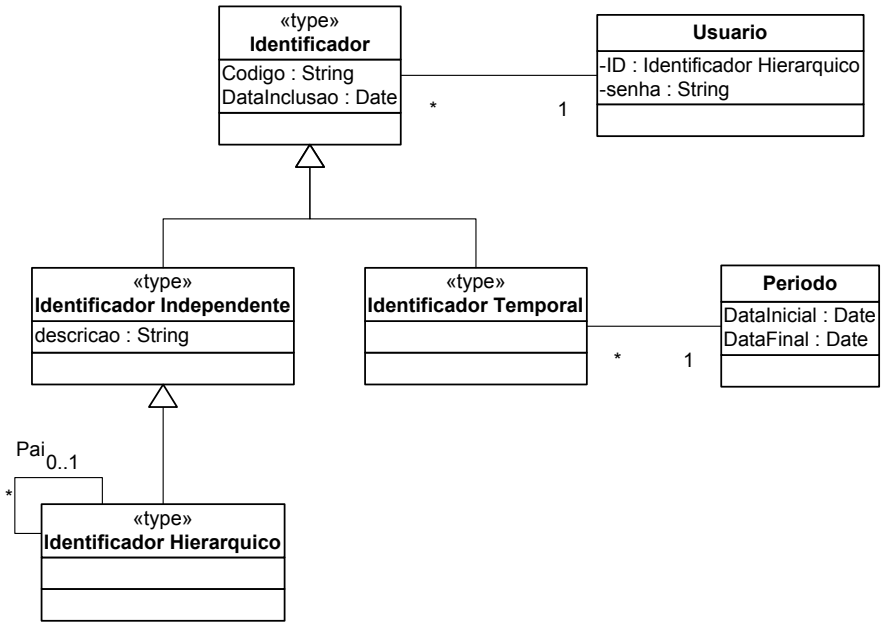

FIGURA 12 - Identificadores

4.2.2.5 Modelo Histórico

Optou-se pela utilização de um modelo simples (FIGURA 13) para armazenar o histórico das transações, privilegiando a manutenção e performance do modelo. O dado que está sendo apagado ou excluído do sistema deverá ser armazenado no atributo **TextoXML**, o qual armazenará os mesmos no formato XML. A opção pelo formato XML se deve ao fato do mesmo ser um padrão largamente suportado e que possibilita a leitura de seu conteúdo sem a necessidade de uma ferramenta específica (HAROLD, 2001).

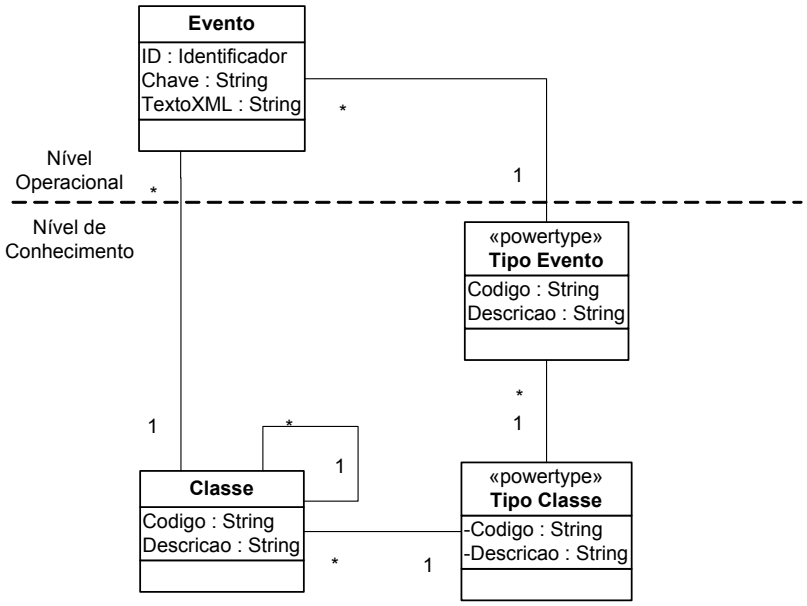

FIGURA 13 - Modelo do Histórico

No modelo do histórico (FIGURA 13) a classe TIPO CLASSE, identifica se a CLASSE é uma 'tela', um 'relatório' ou uma 'tabela do banco de dados', como exemplo. A classe CLASSE, identifica todas as classes existentes no sistema, sobre as quais queremos manter informações históricas e ou controlar o acesso sobre a mesma, tais como 'Relatório de Vendas', 'Tela de Pedidos', 'Tabela de Recursos' entre outras (mais detalhes no item Controle de Acesso); a classe TIPO EVENTO, possui os eventos válidos para as classes, como por exemplo 'inclusão', 'alteração', 'impressão', 'processamento', 'erro', entre outros; e a classe EVENTO, armazena o histórico, propriamente dito; todos os tipos de eventos disparados pelos usuários nas CLASSES podem ser armazenados na classe EVENTO. As classes TIPO EVENTO, CLASSE e TIPO CLASSE não estão associadas à classe EMPRESA e nem à classe USUÁRIO, as mesmas modelam objetos do próprio sistema e não objetos do mundo real, não importa qual EMPRESA estiver utilizando o mesmo, estas classes serão iguais para todas as EMPRESAS e somente serão modificadas pelos desenvolvedores do software e não pelos usuários.

A opção pela não utilização do modelo genérico de histórico (FIGURA 4) se deve à necessidade de constantes atualizações da classe PROPRIEDADE, após cada alteração dos atributos das classes do sistema e também por ser necessário efetuar uma

inserção na tabela de DETALHES DO EVENTO para cada coluna alterada na tabela principal, ocasionando uma degradação da performance.

Na FIGURA 14 têm-se o modelo persistente do 'Histórico', sendo que uma forma de implementá-lo, nos bancos de dados relacionais, é através da criação de gatilhos (*triggers*) de atualização ("update") e deleção ("delete") associados a todas as tabelas do Banco de Dados, desta forma se assegura que sempre será criado o EVENTO associado às alterações das tabelas do banco de dados.

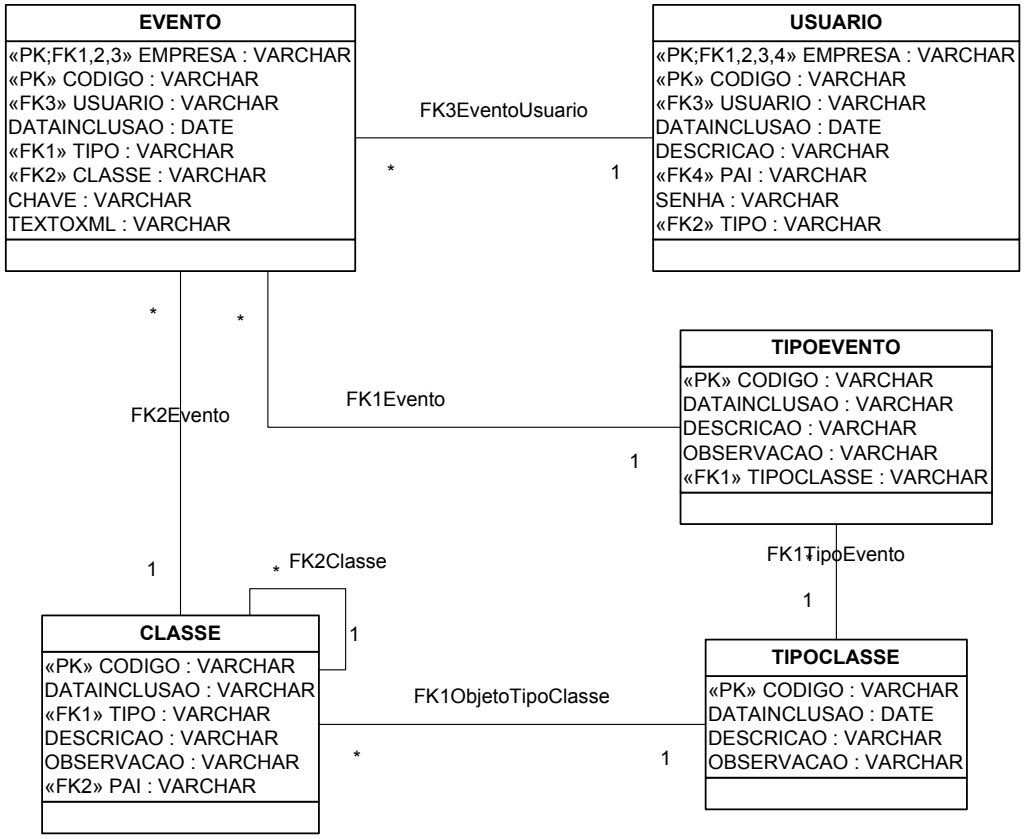

FIGURA 14 - Modelo Persistente do Histórico

Salienta-se que, no histórico podem ser armazenadas não somente informações referentes às atualizações e exclusões do banco de dados, mas também informações sobre erros e estatísticas de utilização e performance de todas as classes do sistema.

4.2.2.6 Modelo para o Controle de Acesso sobre as classes e eventos

O controle de acesso ao sistema é estruturado em vários níveis, se adequando às características de cada modelo. No modelo Histórico, o controle de acesso se dá

pela imposição de restrições ao TIPO USUÁRIO em poder acessar as CLASSES e TIPOS EVENTOS. A classe DIREITO (FIGURA 15 e FIGURA 16) é responsável por liberar o acesso à CLASSE e TIPO EVENTO para um TIPO USUARIO. Como exemplo, podemos ter o objeto 'Vendedor' na classe TIPOUSUÁRIO associado ao objeto 'Cadastro de Cliente' da classe **CLASSE** e ao objeto 'Visualização' da classe **TIPOEVENTO**, desta forma está sendo liberado o acesso de 'Visualização' do 'Cadastro de Cliente' a todos os 'Vendedores'.

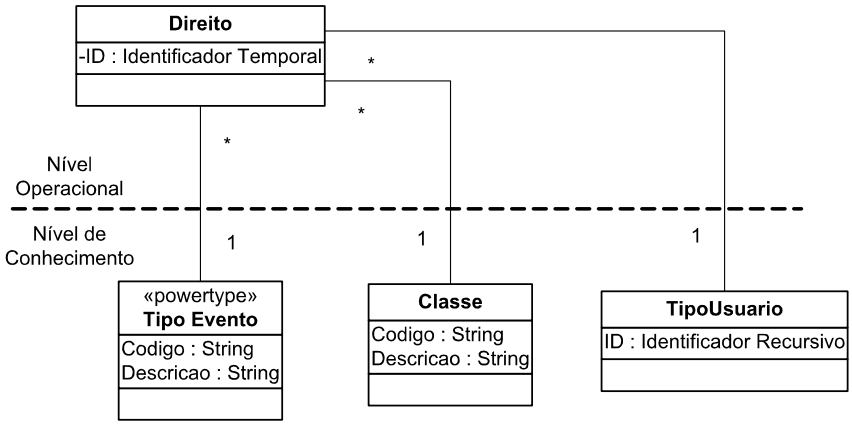

FIGURA 15 - Modelo do Controle de Acesso às classes e eventos

DIREITO
«PK;FK1,2,3,4» -EMPRESA : VARCHAR «PK» -CODIGO : VARCHAR «FK1» -USUARIO : NUMBER -DATAINCLUSAO : DATE -DATAINICIO : DATE -DATAFIM : DATE «FK2» -TIPOUSUARIO : VARCHAR «FK3» -TIPOEVENTO : VARCHAR «FK4» -CLASSE : VARCHAR

FIGURA 16 - Modelo persistente da classe Direito

4.3 Unidades

4.3.1 Padrão Unidade

Normalmente, os computadores armazenam dados sobre objetos do mundo real. Usualmente, estes são modelados como atributos da classe principal, como pode ser visto na FIGURA 17. Neste caso, se João tem seu peso igual a 80 Kg, pode-se

armazenar este dado no atributo **Peso** da classe PESSOA. Apesar desta forma ser a mais tradicional, a mesma apresenta problemas (FOWLER, 1997). Se for armazenado um número, que representa o peso da pessoa, não se sabe o peso da mesma somente através deste número. Se 'João' pesa '120' ou '6', não sabemos quanto 'João' pesa, necessitamos saber as unidades de medidas associadas a estes valores.

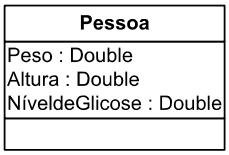

FIGURA 17 - Forma típica para armazenamento de informações sobre um objeto (FOWLER, 1997)

Para resolver este problema, pode-se colocar na descrição do atributo a unidade, tal como **PesoEmKg**. Desta forma, não se estaria armazenando, necessariamente, o dado em sua forma original de medida. Em alguns sistemas, como os ligados à medicina, necessita-se armazenar o dado juntamente com a unidade utilizada na medição, para que a mesma represente fielmente a realidade (FOWLER, 1997).

Neste contexto, pode-se utilizar o tipo **QUANTIDADE** (FIGURA 18), o qual combina o **Valor** com sua **UnidadeMedida** (FOWLER, 1997). Ao modelar a **UnidadeMedida** como sendo do tipo **UNIDADE** (FIGURA 19), facilita-se a criação de métodos para conversão entre unidades. Para tanto, pode-se utilizar a classe TAXA DE CONVERSÃO (FIGURA 19) que possibilita criar os métodos de conversão entre UNIDADES diferentes. Como exemplo, pode-se converter pés em polegadas definindo uma taxa de conversão (FOWLER, 1997). Este modelo não funciona para casos mais complexos como converter graus Celsius para Fahrenheit ou ainda meses em dias, pois estas conversões não são resultado de multiplicações simples, necessitando métodos individuais de tratamento (FOWLER, 1997).

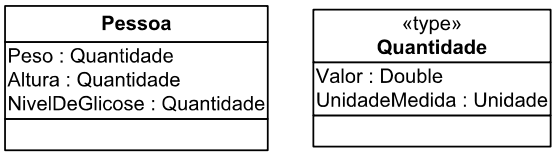

FIGURA 18 - Padrão Quantidade (FOWLER, 1997)

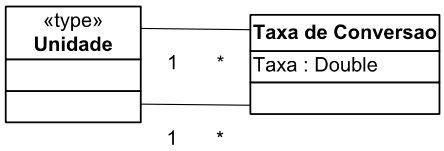

FIGURA 19 – Padrão Unidade (FOWLER, 1997)

Salienta-se que os valores monetários também podem ser representados por este padrão, sendo a moeda representada pela **UnidadeMedida** do **Valor** . Para que os métodos de conversão funcionem neste caso, necessita-se adicionar à classe TAXA DE CONVERSÃO um atributo do tipo **Período de Tempo** que serve para determinar a validade da taxa, pois a taxa de conversão entre as moedas variam com o passar do tempo (FOWLER, 1997).

4.3.2 Modelo Unidades

As unidades físicas e monetárias utilizadas no sistema como ‘R$’ (Reais) , ‘Us$’ (Dólar), ‘Kg’ (Kilograma), entre muitas outras são UNIDADES. Para facilitar o cadastro das UNIDADES, seu **Id** é do tipo **IDENTIFICADOR HIERÁRQUICO** (FIGURA 20) tornando possível a classificação das unidades e auxiliando na melhor usabilidade do sistema. Já a classe CONVERSÃO serve para associar duas unidades a uma taxa de conversão, a qual será utilizada na conversão de uma unidade para a outra. Salienta-se a presença do atributo **Validade** que é do tipo **PERÍODO DE TEMPO** e que serve para determinar o período de validade da taxa de conversão, característica esta muito utilizada com as unidades monetárias. Como exemplo de utilização da classe CONVERSÃO podemos ter que ‘um Dólar’ vale ‘2,31’ ‘Reais’

entre ‘30/06/01 00:00’ a ‘31/06/01 00:00’. Na FIGURA 21 pode se visualizar o modelo persistente dos objetos CONVERSÃO e UNIDADE.

Salienta-se que os atributos que objetivam armazenar valores monetários ou quantidades físicas no sistema são todos do tipo **QUANTIDADE (**FIGURA 20**)** , que é composto pelo atributo **Valor** e pelo relacionamento com a classe UNIDADE. No modelo relacional as colunas que armazenam valores ou quantidades são sempre acompanhadas da coluna que contém a sua unidade.

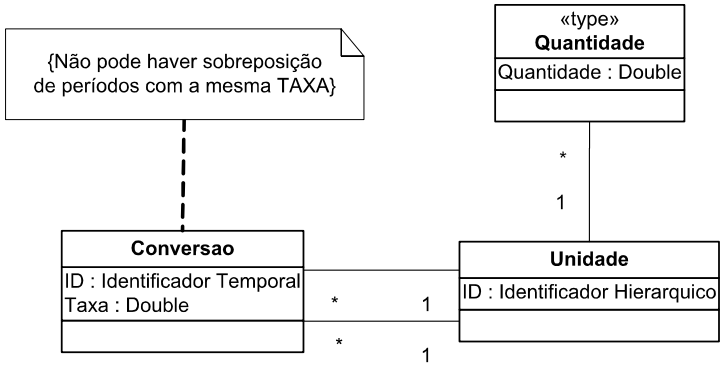

FIGURA 20 – Tipo Quantidade

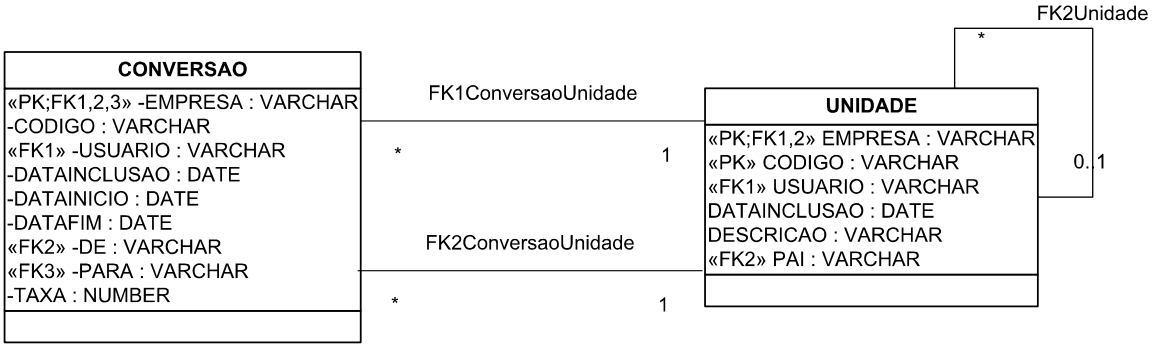

FIGURA 21 - Modelo Persistente das Unidades

4.4 Recursos

> "Recursos são os objetos que atuam ou são utilizados pelos negócios" **(ERIKSSON; PENKER, 2000).**

4.4.1 Padrão Responsável

A estruturação hierárquica das informações é uma prática comum nas empresas, nestas encontra-se estruturas hierárquicas de pessoas, unidades de negócio, materiais, contas contábeis entre outras (FOWLER, 1997), estas estruturas podem ser representadas de diversas formas as quais serão analisadas neste item.

O modelo da estrutura de uma empresa pode ser feito de forma tradicional, como mostrado no diagrama da FIGURA 22. Este diagrama representa uma empresa que pode ser formada por várias unidades de negócio, as quais são subdivididas em regiões, divisões e escritórios. O problema com este tipo de modelagem é que se a empresa resolver retirar ou adicionar um dos seus níveis organizacionais, o modelo terá que ser refeito, ou seja, o modelo não é flexível às mudanças no negócio (FOWLER, 1997).

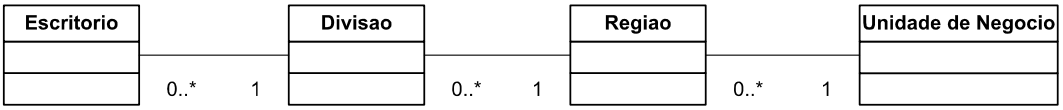

FIGURA 22 – Modelagem Tradicional da estrutura organizacional de uma empresa

Na FIGURA 23, temos o padrão Hierarquia Organizacional (do original *Organization Hierarchy*) (FOWLER, 2000d), um modelo recursivo simples que suporta mudanças organizacionais sem a necessidade de alterações no modelo. O problema deste padrão é que o mesmo utiliza um relacionamento recursivo, o qual permite a um Escritório estar ligado diretamente a uma Unidade de Negócio ou até a si mesmo, possibilitando inconsistências, fato este que não ocorre no diagrama tradicional (FIGURA 22) (FOWLER, 1997).

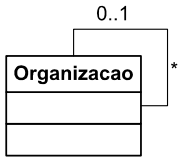

FIGURA 23 – Padrão Hierarquia Organizacional (*Organization Hierarchy* ; FOWLER, 2000D)

Para resolver o problema do diagrama recursivo, pode-se adicionar a classe supertipo TIPO DA ORGANIZAÇÃO (FIGURA 24). A mesma determinará os tipos de organizações válidas e possibilitará a criação de restrições que consistirão a estrutura hierárquica da classe ORGANIZAÇÃO, verificando se a mesma segue o modelo estabelecido na hierarquia da classe TIPO DA ORGANIZAÇÃO. Para isso, basta verificar se a estrutura da ORGANIZAÇÃO é a mesma da estrutura do TIPO DA ORGANIZAÇÃO. Neste modelo, bastam que sejam atualizados os objetos das duas classes para adaptação a uma nova estrutura da empresa (HAY, 1996).

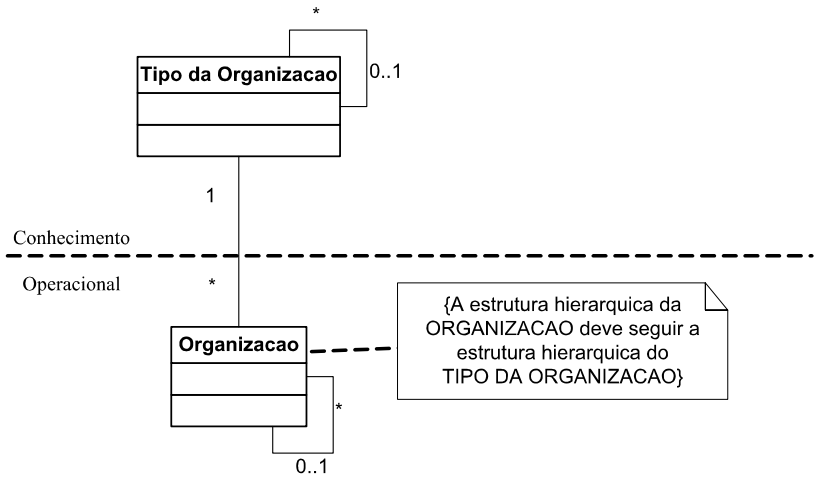

FIGURA 24 - Padrão Hierarquia Organizacional com supertipo

Apesar do modelo hierárquico com super-tipo (FIGURA 24) ser bastante flexível, o mesmo suporta uma hierarquia simples, a qual não permitiria que uma loja se reportasse a duas estruturas superiores simultaneamente. O modelo com duas hierarquias, da FIGURA 25, resolve este problema ao adicionar mais uma estrutura hierárquica. Esta não é uma boa solução, pois o número de hierarquias permitidas ainda estaria restrito. Caso fossem necessárias mais hierarquias, o modelo se tornaria

pesado, além de tornar a modelagem dos subtipos e restrições mais complexa. (FOWLER, 1997).

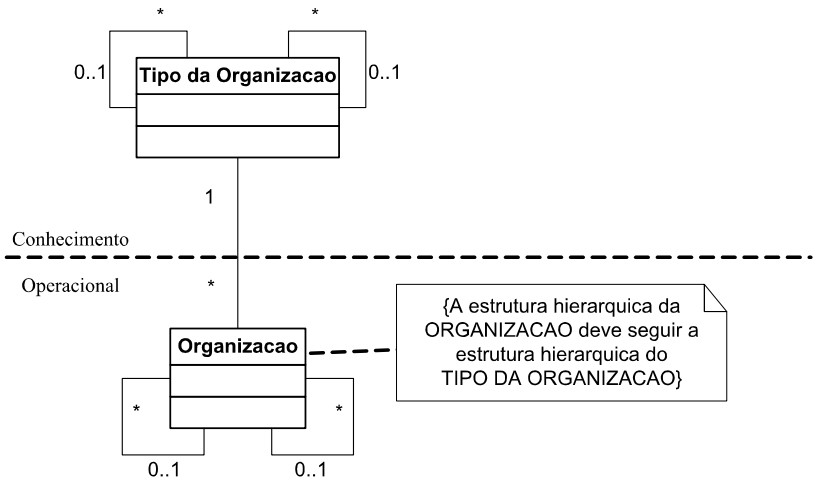

FIGURA 25 - Padrão Hierarquia Organizacional com duas hierarquias

Para melhorar esta representação, foram transformados os auto-relacionamentos em classes de relacionamento com um supertipo e os nomes das classes foram alterados para tornar o modelo mais genérico. Como pode ser visto na FIGURA 26, a classe ORGANIZAÇÃO passou a se chamar GRUPO (do original "Party") e o TIPO DA ORGANIZAÇÃO passou a se chamar TIPO DO GRUPO (do original "Party Type"). Cada instância da classe RESPONSÁVEL (do original "Accountability") representa uma ligação entre dois GRUPOS e o TIPO RESPONSÁVEL (do original "Accountability Type") indica a natureza das ligações. Esta estrutura permite trabalhar com qualquer número de hierarquias, bastando criar uma instância na classe TIPO RESPONSÁVEL e ligar as organizações entre si (FOWLER, 2000d). De forma análoga ao modelo anterior, deve-se criar restrições que somente permitam criar objetos na classe RESPONSÁVEL se o mesmo seguir o modelo estabelecido na classe REGRAS DE CONEXÃO (do original "Connection Rule"). Este modelo cria uma estrutura flexível com hierarquias múltiplas que permitem diferentes visualizações de um mesmo negócio (FOWLER, 2000d).

No diagrama da FIGURA 24 e nos subseqüentes, é salientada a separação entre o nível operacional e o nível do conhecimento. O nível operacional é responsável por registrar os eventos do dia a dia, enquanto o do conhecimento armazena as regras

gerais que governam a estrutura (FOWLER, 1997). Tem-se um nível de conhecimento, quando um grupo de objetos e suas instâncias afetam o comportamento dos objetos do nível operacional. Tipicamente, alterando-se o nível de conhecimento se está alterando o sistema, sem necessitar alterar a programação do mesmo. Desta forma, seria possível ao usuário final, alterar os objetos do nível de conhecimento e assim alterar o comportamento do sistema, mas este fato ainda é um sonho, dada a complexidade do nível do conhecimento, o mesmo normalmente é alterado por programadores experientes (FOWLER, 2000b). A linha pontilhada que separa o nível de conhecimento do operacional não é um padrão da UML (FOWLER, 2000b).

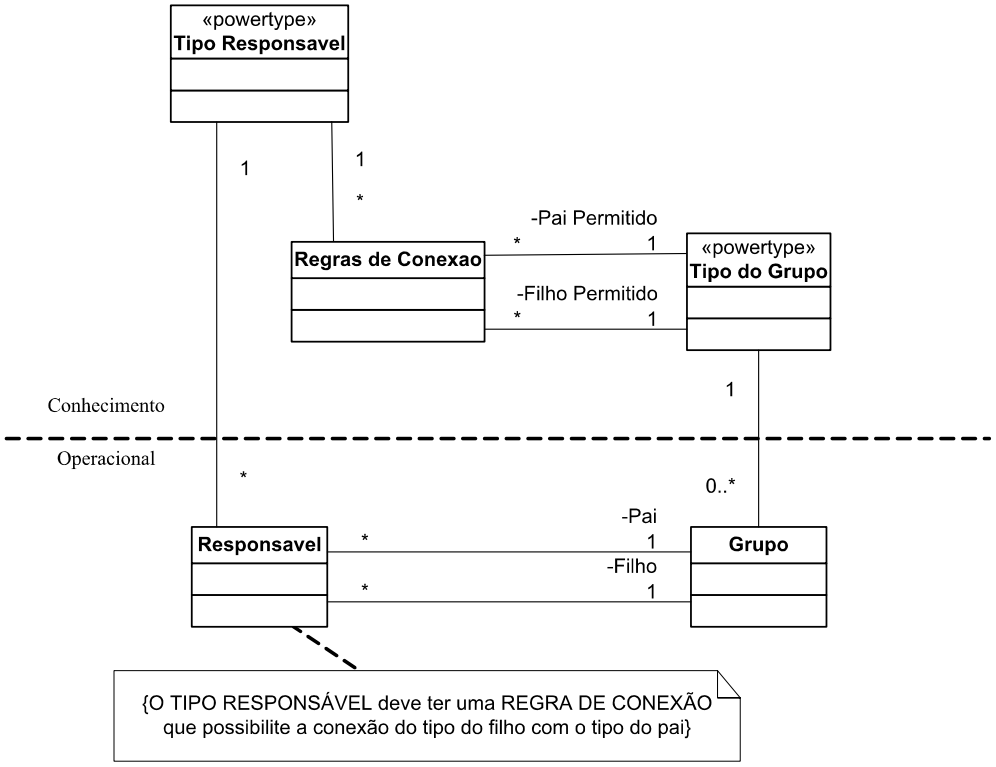

FIGURA 26 – Padrão Responsável (*Accountability*; FOWLER, 2000D)

4.4.2 Modelo Recursos

O modelo Recursos se baseia no padrão Responsável (do original *Accountability)* (FIGURA 26) sendo esta uma forma bastante genérica para se modelar uma estrutura hierárquica dentro do sistema. Apesar deste modelo ter sido originalmente concebido para modelar a estrutura organizacional de uma empresa, neste trabalho o mesmo será utilizado de forma mais genérica. Os objetos 'produtos',

‘clientes’, ‘contas’, ‘médicos’, ‘convênios’, ‘fornecedores’, ‘empregados’, ‘unidades organizacionais’, entre outros, são exemplos de RECURSOS. Na verdade, os objetos que atuam ou são utilizados pelos negócios e que não são DOCUMENTOS são considerados pelo sistema como RECURSOS, podendo ser físicos, abstratos ou informacionais.

Salienta-se que o plano de contas contábil em uma empresa é organizado de forma hierárquica. Os materiais, os produtos, fornecedores, clientes, funcionários, entre outras estruturas, também podem ser organizados desta forma, assim sendo, é conveniente que se utilize o padrão Responsável para modelar estas estruturas de forma flexível.

Conforme pode ser visto nos diagramas da FIGURA 27 , têm-se a classe TIPORECURSO, que indicará se um recurso é do tipo ‘médico’, ‘fornecedor’, ‘produto’, ‘cliente’ ou ainda do tipo ‘título’ (tipo abstrato que serve para compor os galhos das árvores), entre outros. Em nosso modelo, optou-se por associar um RECURSO a um único TIPORECURSO, apesar de se reconhecer que, às vezes, um ‘cliente’ pode também ser ‘fornecedor’, por exemplo. Optou-se por esta modelagem em favor da simplificação do modelo e de sua implementação, da mesma forma que no padrão Responsável.

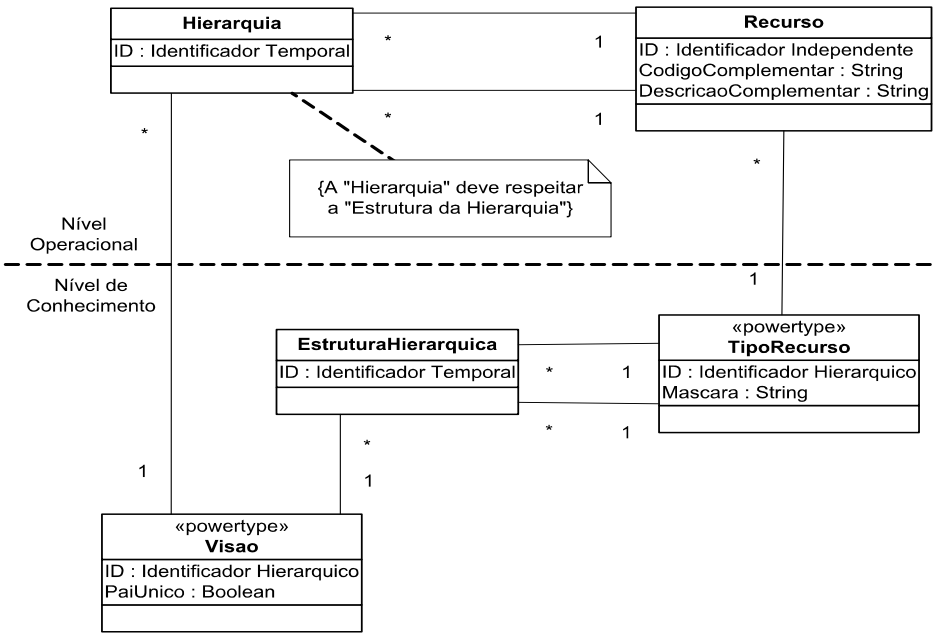

FIGURA 27 - Modelo Recursos

O atributo **Código**, da classe RECURSO, como também em todas as demais classes do sistema, necessariamente deve ser único no escopo do sistema. Esta restrição faz com que não se possa atribuir um mesmo **Código** a objetos de tipos diferentes, como exemplo, não se pode ter o cliente 'Eli S/A' com o **Código** '1010' e o fornecedor 'Azarão Ltda' com o mesmo **Código** (salvo se os mesmos forem utilizados por Empresas diferentes dentro do sistema). Apesar de não ser uma obrigação, normalmente é aconselhado ao cliente que utilize o número seqüencial que o próprio sistema pode propor quando da criação de um novo RECURSO. Apesar disto, o atributo **Código** foi definido como do tipo **STRING** para poder se adequar a uma eventual codificação específica que o cliente já possua em sua empresa e queira manter. Caso o cliente aceite a sugestão de utilizar o código seqüencial, mas seja necessário manter a codificação antiga para algumas operações específicas, o atributo **CodigoComplementar** (que não é obrigatório) pode ser utilizado. O mesmo foi criado com a finalidade de ser um código alternativo do recurso, facilitando a importação e integração dos dados vindos de outros sistemas.

O atributo **Descrição** é utilizado juntamente com o atributo **DescricaoComplementar**, sendo que os mesmos descrevem o RECURSO. Como exemplo, quando o RECURSO é uma pessoa jurídica, o normal é armazenar a sua razão social na **Descrição** e utilizar a **DescriçãoComplementar** para o seu nome fantasia ou sua descrição resumida. Já se o RECURSO for uma pessoa física, atribui-se o primeiro nome e nomes intermediários à **Descrição** e o seu sobrenome à **DescriçãoComplementar**. Esta opção, de se ter duas descrições, facilita a busca dos recursos por parte do usuário, quando necessário.

O modelo Recursos, na verdade, armazena somente o código de cada RECURSO, sua descrição, seu tipo e a sua localização nas estruturas hierárquicas definidas, não armazenando as características dos recursos propriamente ditos. Para exemplificar, ao cadastrar um novo cliente é necessário armazenar vários dados como endereço, CNPJ, tipo de contrato, entre vários outros, e não somente seu código e descrição. Como será visto no modelo Caráter, o mesmo é responsável por armazenar tais informações, adequando as mesmas a cada um dos recursos, conforme a

necessidade do usuário. O atributo **Caráter** é responsável pela interligação entre as classes RECURSO e CARÁTER.

As múltiplas hierarquias que são suportadas pelo modelo foram chamadas de VISÕES. Cada VISÃO no sistema tem a ela associada uma HIERARQUIA distinta de recursos. Estas HIERARQUIAS são necessárias para modelar VISÕES de usuários diferentes sob o mesmo sistema, como exemplo, um contador pode ter uma VISÃO sobre os RECURSOS da empresa de forma diferenciada da VISÃO da área industrial, que pode ser diferente do financeiro e da área de custos.

A ESTRUTURA HIERARQUICA define o TIPO RECURSO que pode ser pai de um TIPO RECURSO filho, ou seja, podemos ter um RECURSO do tipo 'Funcionário' que pode ser filho de um RECURSO do tipo 'Função', que pode ser filho de um recurso do tipo 'Setor', que pode ser filho de um recurso do tipo 'Unidade de negócio', isto na VISÃO 'Principal'. Já na VISÃO 'Organograma', pode-se ter um 'Funcionário' que pode ser filho de um outro 'Funcionário'. Estas regras, determinadas pela classe ESTRUTURA HIERARQUICA, devem ser respeitadas quando forem criados os objetos da classe HIERARQUIA, que será a classe que determinará a estrutura hierárquica dos RECURSOS em cada VISÃO.

Como a HIERARQUIA é uma estrutura dinâmica, ou seja, um funcionário que trabalha em um setor pode ser transferido para outro, ou pode ser demitido, estes eventos afetam os objetos da HIERARQUIA, fazendo com que a mesma necessite determinar o tempo de validade de uma determinada relação. Por este motivo, necessita-se do **IDENTIFICADOR TEMPORAL**, que serve para determinar por quanto tempo a relação foi, será, ou começará a ser válida.

Já no modelo Estrutura da Hierarquia, é necessário o **IDENTIFICADOR TEMPORAL** para determinar o final de uma estruturação e o início de uma nova. Salienta-se que, ao alterar a ESTRUTURA HIERARQUICA é necessário reorganizar todos os elementos que são afetados pela mudança na HIERARQUIA, para manter o sistema íntegro.

Na FIGURA 28 pode-se observar o modelo persistente dos Recursos que será armazenado em um banco de dados relacional.

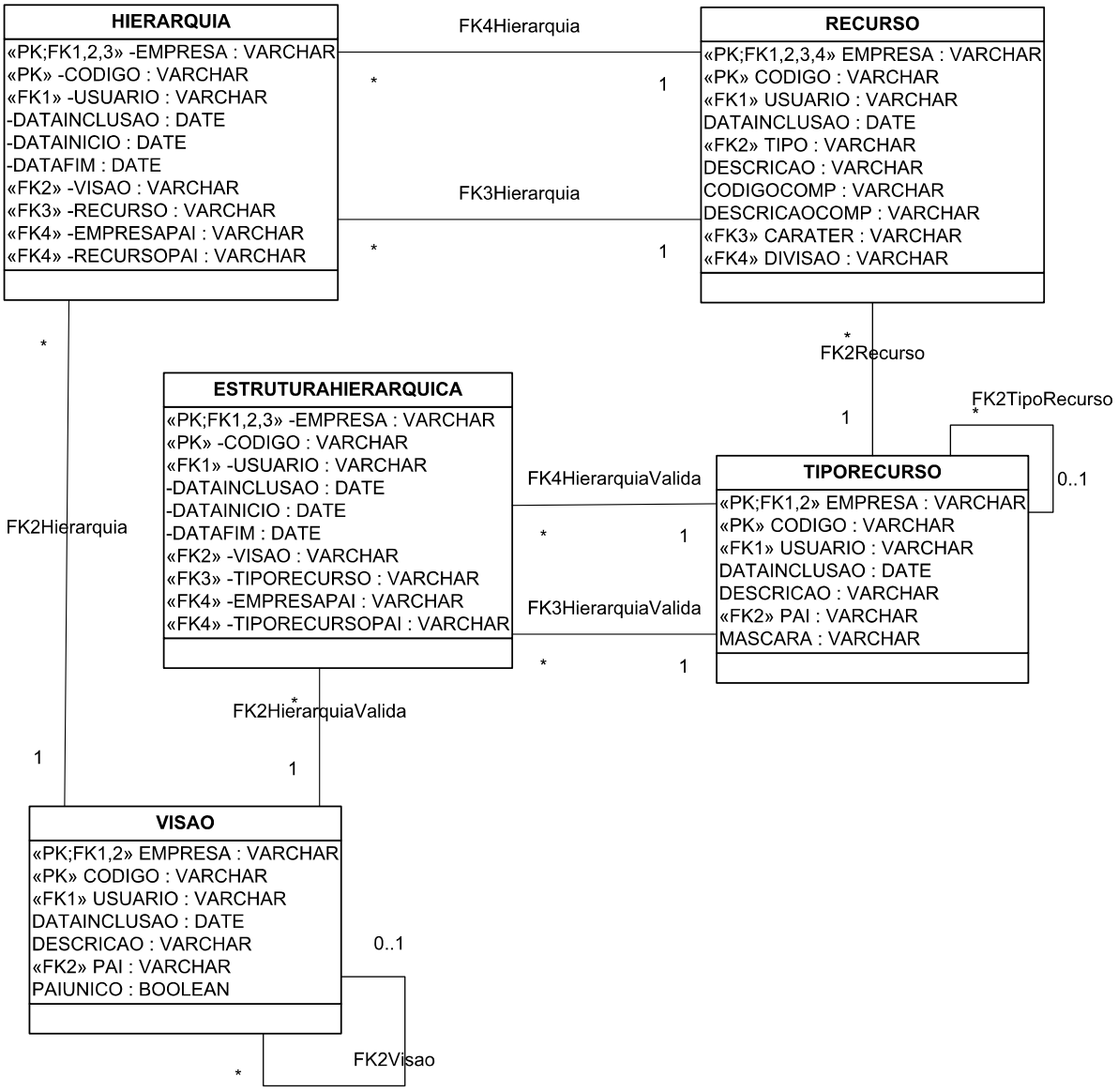

FIGURA 28 – Modelo Persistente dos Recursos

4.4.2.1 Controle de acesso aos recursos

Se for criada mais de uma EMPRESA no sistema, estas terão independência completa em relação aos seus dados. Se forem criadas DIVISÕES dentro de uma EMPRESA, os dados não serão independentes entre as DIVISÕES, mas pode-se restringir o acesso a dados específicos de uma DIVISÃO em relação a outra. Já se for

necessário que uma DIVISÃO tenha exatamente os mesmos acessos que outra DIVISÃO esta será uma subdivisão da mesma.

Se for necessário que alguns RECURSOS de uma DIVISÃO não sejam acessados pelas demais, pode-se associar aos mesmos uma DIVISÃO específica, restringindo assim o acesso das demais DIVISÕES a este RECURSO (FIGURA 29). Por outro lado, se o RECURSO for compartilhado entre todas DIVISÕES, basta não associar ao RECURSO nenhuma DIVISÃO.

Num sistema hospitalar, para exemplificar, poder-se-ia ter as seguintes DIVISÕES: 'Hospital', 'Laboratório', 'RX' e 'Posto de Coleta'. Deseja-se que as DIVISÕES compartilhem o cadastro de Clientes, Médicos e Convênios e sejam independentes em relação ao cadastro de seus produtos, ressaltando somente que o 'Posto de Coleta' é uma sub-Divisão externa ao 'Hospital' a qual recebe materiais para exames no 'Laboratório'. Assim sendo, deve ser criada no sistema uma empresa única com quatro DIVISÕES, 'Hospital', 'Laboratório', 'RX' e 'Posto de Coleta', sendo que o 'Posto de Coleta' será filho do 'Laboratório', ou seja uma Sub-Divisão do mesmo. Os RECURSOS 'Clientes', 'Médicos' e 'Convênios' não serão associados a nenhuma DIVISÃO, já os RECURSOS 'Produtos' deverão ser associados a respectiva DIVISÃO que é responsável pelo mesmo, a DIVISÃO 'Posto de Coleta', por ser uma sub-Divisão, não terá nenhum RECURSO associado a si, mas herdará os mesmos da DIVISÃO 'Laboratório'.

Além de se poder restringir o acesso a alguns RECURSOS, conforme a DIVISÃO, pode-se também restringir o acesso conforme o TIPO USUÁRIO a certo TIPO RECURSO específico. Salienta-se que o usuário ao entrar no sistema (*log-in*) deverá sempre informar a empresa, a divisão, sua identificação e senha. Assim sendo, quando o mesmo for acessar a tabela de RECURSOS o sistema verificará quais são os TIPO RECURSO que o usuário tem direito de acessar na DIVISÃO especificada e filtrará o acesso aos mesmos para o usuário. A classe DIREITO TIPO RECURSO é responsável por liberar o acesso de um TIPO USUÁRIO para um TIPO RECURSO em uma DIVISÃO determinada (FIGURA 29), A FIGURA 30 apresenta a tabela

DIREITOTIPORECURSO que é o resultado da conversão da respectiva classe para o modelo relacional.

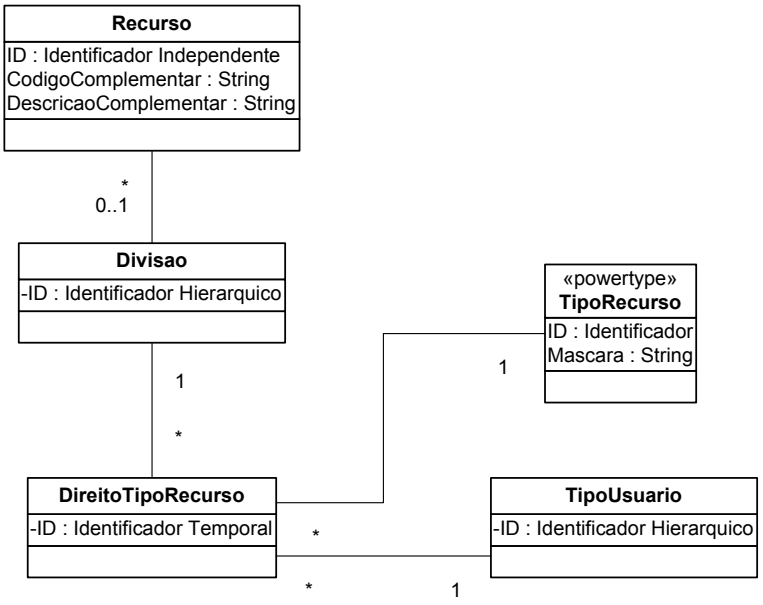

FIGURA 29 - Controle de acesso aos recursos

DIREITOTIPORECURSO
«PK;FK1,2,3,4» -EMPRESA : VARCHAR
«PK» -CODIGO : VARCHAR
«FK1» -USUARIO : NUMBER
-DATAINCLUSAO : DATE
-DATAINICIO : DATE
-DATAFIM : DATE
«FK2» -TIPOUSUARIO : VARCHAR
«FK3» -TIPORECURSO : VARCHAR
«FK4» -DIVISAO : VARCHAR

FIGURA 30 - Classe persistente Direito Tipo Recurso

4.5 Caráter

"Caráter: traço distintivo de uma pessoa ou coisa" (HOUAISS, 2001).

4.5.1 Padrão observações

No tópico Unidades foi mostrado a forma típica que se armazena informações em sistemas computadorizados (FIGURA 17), esta forma, além de possuir os problemas associados com as unidades de medidas, também não é flexível. Sempre que for necessário adicionar um novo dado a um conceito já definido, terá de ser feita

uma mudança no modelo com a respectiva adição de um novo atributo a classe e proceder com todas as manutenções, decorrentes deste ajuste, ou seja, será uma alteração demorada.

A dinamicidade dos negócios atualmente não permite mais que sistemas demorem demasiadamente para se adaptar a uma nova necessidade de informações, além de cada vez mais as empresas estarem negociando com clientes diferenciados com necessidades específicas de informações e produtos. As informações também estão se tornando específicas a novas necessidades.

Para exemplificar, num sistema hospitalar um paciente possui características importantes a serem armazenadas, tais como: pressão, peso, temperatura, grupo sanguíneo, se é diabético, se é fumante, entre outras. Modelar estes dados como atributos da classe PESSOA pode ser útil para um departamento de um hospital que coleta poucos dados sobre seus pacientes, mas sob uma visão mais ampla de um hospital e da medicina que lá se aplica, serão encontrados milhares de dados que podem ser medidos em uma pessoa. Se for definido um atributo para cada dado, ter-se-á uma classe com uma interface complexa, uma vez que a mesma pode necessitar de milhares de atributos e alterações constantes, pois a cada dia novas técnicas surgem e mais informações sobre o paciente são necessárias (FOWLER, 1997).

Para resolver este problema, FOWLER (1996) propôs que as várias medidas possíveis (pressão, peso, temperatura, grupo sanguíneo, entre outros) sobre o objeto PESSOA, fossem consideradas como sendo TIPO DE FENÔMENO (FIGURA 31) e que o mesmo se relacionasse com a classe PESSOA através da classe MEDIDA a qual seria responsável por armazenar a **Quantidade** (composição de um valor e sua respectiva unidade) do TIPO DE FENÔMENO associado a PESSOA. Desta forma, cada PESSOA pode ter tantas MEDIDAS quantas forem necessárias dos TIPOS DE FENÔMENOS desejados pelo médico, por exemplo. Podendo-se ainda ter atributos associados à classe MEDIDA que indicam quando foi feita a medida, por quem e onde, entre outros (FOWLER, 1997).

Para exemplificar, se o objeto ‘João’ mede 1,7 metros e pesa 80 Kg, tem-se que, o objeto ‘João’ é uma PESSOA seu ‘peso’ e a ‘altura’ são os TIPOS DE FENÔMENO e possui MEDIDAS de ‘1,7 metros’ e ‘80 Kg’.

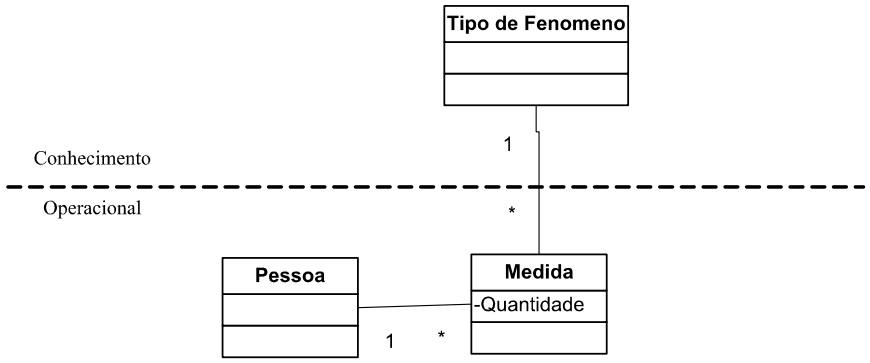

FIGURA 31 – Padrão Medida (*Measurement*; FOWLER,1996)

Um dos problemas deste modelo está no fato de que nem todas as informações são compostas por números com sua unidade de medida. Muitas vezes é necessário armazenar informações do tipo ‘positivo’ / ‘negativo’ ou ‘homem’ / ‘mulher’, entre outras. Para se resolver este problema, foi criada a classe OBSERVAÇÃO (FIGURA 32), a qual terá duas especializações: a MEDIDA e a CATEGORIA DA OBSERVAÇÃO, que determinará o FENÔMENO da OBSERVAÇÃO. Para finalizar, é necessário criar uma classe que informe quais são os fenômenos válidos para um TIPO DE FENÔMENO, sendo esta classe o FENÔMENO (FOWLER, 1997).

Para exemplificar, pode-se ter os TIPO DE FENÔMENO ‘sexo’ e ‘tipo sangüíneo’, que possuem respectivamente os FENÔMENOS ‘masculino’, ‘feminino’ e ‘A’, ‘B’, ‘AB’, ‘O’; assim sendo, uma PESSOA em uma OBSERVAÇÃO pode ser do ‘sexo’ ‘masculino’ e ter o ‘tipo sangüíneo’ ‘A’, sendo estas informações válidas por um **PERÍODO DE TEMPO** determinado.

Em um sistema de estoque industrial, o problema é similar, pois é necessário armazenar informações específicas sobre cada item em estoque. Exemplificando, podem-se ter resinas plásticas com informações de fluidez, cor e resistência. Já um parafuso, necessita de outros tipos de informações, como tipo de rosca, tamanho e peso. Salienta-se que este modelo também se adapta a esta necessidade.

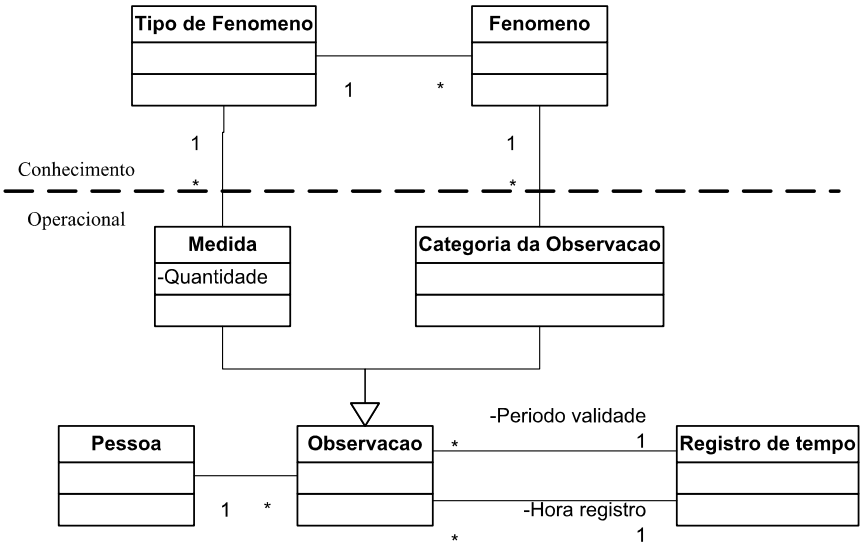

FIGURA 32 - Padrão observações (*Observation*; FOWLER,1996)

4.5.1.1 Observações rejeitadas

Inevitavelmente, ao serem feitas observações, podem-se cometer enganos e, no caso de registros médicos, não se pode simplesmente apagar os mesmos. Tratamentos podem ser aplicados baseados em informações errôneas tendo conseqüências legais decorrentes deste erro. Para tratar tal fato podem-se classificar essas OBSERVAÇÕES como OBSERVAÇÕES REJEITADAS (FIGURA 33) (FOWLER, 1997). Deve-se distinguir claramente a diferença entre uma OBSERVAÇÃO REJEITADA e uma observação que não é mais válida. A observação é rejeitada em decorrência de um dado errado, já uma observação não válida hoje, pode ter sido ou será válida no período de tempo especificado para a mesma (FOWLER, 1997).

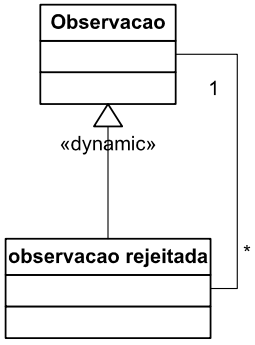

FIGURA 33 – Padrão Observações rejeitadas (FOWLER,1996)

4.5.1.2 Observações ativas, hipóteses e projeções

Ao ser feita uma OBSERVAÇÃO, a mesma pode ter vários níveis de confiabilidade. Quando um médico está diante de um paciente com sintomas de diabete, o médico registra que provavelmente o seu paciente tenha diabetes enquanto providencia exames mais precisos que comprovem o seu diagnóstico, ou seja, isto é uma hipótese. A OBSERVAÇÃO de 'diabetes' passará a ser ativa quando a mesma for efetivamente comprovada com os exames.

Outra situação é quando um paciente tem, por exemplo, uma febre reumática que poderá causar no futuro problemas cardíacos. Esta é uma PROJEÇÃO que merece ser considerada para que se tenha o efetivo tratamento. A FIGURA 34 mostra o modelo que classifica estas características.

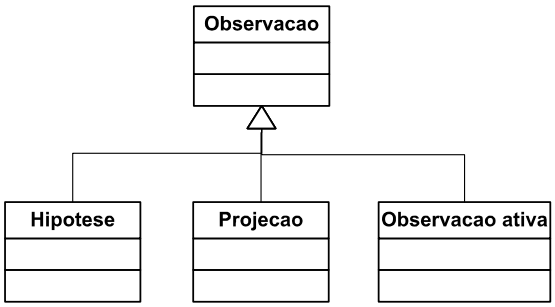

FIGURA 34 - Padrão das Hipóteses, Projeções e Observações Ativas
(FOWLER,1996)

4.5.2 Padrão Universal de Dados

HAY (1996), propôs o Padrão Universal de Dados (FIGURA 35), o qual "serve para modelar Pessoas, Ordens de Produção, Materiais e todas as coisas do universo, ou quase todas" HAY (1996). Este modelo é resultado da observação de vários modelos, os quais possuíam estrutura específica e foram generalizados neste modelo (HAY, 1996).

No Padrão Universal de Dados (HAY, 1996) o mundo é composto pela classe OBJETO, que possui as informações de suas propriedades armazenadas na classe VALOR. A estrutura do OBJETO é definida pela classe TIPOOBJETO, sendo que para cada uma podemos ter um ou vários ATRIBUTOS.

Cada OBJETO está associado a um ou mais OBJETOS através da classe RELACIONAMENTO, sendo que cada RELACIONAMENTO entre os OBJETOS deve ser do tipo TIPO DO RELACIONAMENTO. Pode-se desta forma criar estruturas hierárquicas múltiplas entre os OBJETOS (HAY, 1996).

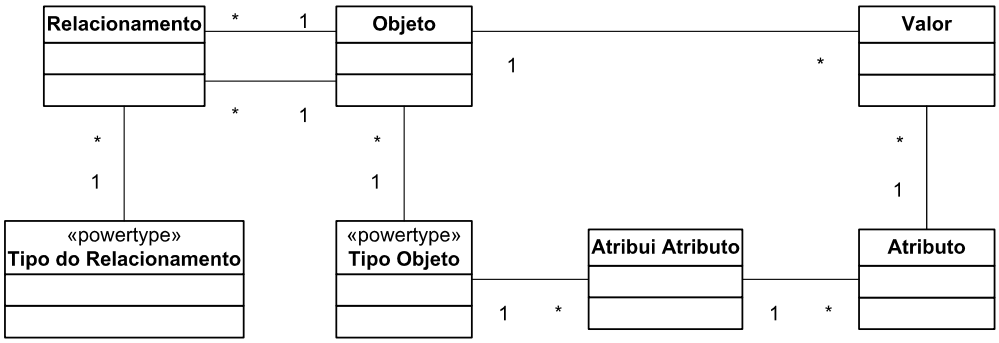

FIGURA 35 - Modelo de Dados Universal (HAY, 1996)

O Padrão Universal de Dados não se restringe a uma classe específica como PESSOA, o mesmo possui a classe TIPO OBJETO a qual pode ser uma 'Pessoa', 'Material', 'Conta', entre outras. O TIPO OBJETO é constituído por vários **Atributos** (através da classe ATRIBUI ATRIBUTO*)* que, com seus **Valores** associados, caracterizam um determinado OBJETO. Para exemplificar, imagine que se tenha os objetos 'Pessoa' e 'Parafuso' na classe TIPO OBJETO; os objetos 'Peso', 'Altura' e 'Tipo de cabeça' na classe ATRIBUTOS; e que à 'Pessoa' relacionamos os ATRIBUTOS 'Peso' e 'Altura' e aos 'Parafusos' relacionamos os ATRIBUTOS 'Peso' e 'Tipo de cabeça'. Assim, pode-se ter que 'João' (OBJETO), que é uma 'Pessoa' (TIPOOBJETO), 'Pesa' (ATRIBUTO) '*80 Kg'* (VALOR) e sua 'Altura' (ATRIBUTO) é de '1,7 metros' (VALOR). Como o 'Tipo de Cabeça' não é um ATRIBUTO da TIPO OBJETO 'Pessoa', não se pode dizer que 'João' (OBJETO) tem um 'tipo de cabeça' (ATRIBUTO), 'sextavada' (VALOR), pois o 'tipo de cabeça' é um ATRIBUTO associado ao TIPO OBJETO 'Parafusos' e não 'Pessoa'.

Como se pode observar, a lógica dos modelos propostos por FOWLER e por HAY são parecida, sendo o Padrão Universal de Dados mais genérico do que o Padrão Observações. O modelo proposto a seguir se baseia em ambos.

4.5.3 Modelo Caráter

Numa abordagem tradicional, ao modelar um sistema laboratorial, por exemplo, ter-se-ia as classes MEDICO, PACIENTE, CONVENIO, EXAME entre outras, com seus atributos pré-definidos. Neste trabalho, estas classes não existem, os objetos que as mesmas conteriam são, na verdade, objetos da classe RECURSO e os seus atributos são definidos pelo modelo Caráter.

Fazendo uma comparação entre o modelo Padrão Universal de Dados e o Modelo Recursos, têm-se as classes OBJETO, TIPODORELACIONAMENTO e RELACIONAMENTO do Padrão Universal de Dados que equivalem às classes RECURSOS, VISÃO e HIERARQUIA no Modelo dos Recursos. Como se pode notar, neste trabalho optou-se por separar o modelo Recurso do Caráter, tornando ambos independentes em relação a suas identificações, ou seja, no modelo relacional as tabelas de ambos possuem chaves primárias independentes um do outro. Esta opção surgiu da necessidade de ter-se VALORES de PROPRIEDADES para objetos do tipo RECURSO como também para objetos do tipo DOCUMENTOS, que serão vistos no próximo item. Como exemplo, no modelo desenvolvido neste trabalho, as PROPRIEDADES 'data de nascimento', 'peso', 'telefone', entre outros, podem estar associadas a um 'Cliente', como também se pode ter as PROPRIEDADES '% ICMS', 'Classificação Fiscal', 'Peso Bruto das Mercadorias', associadas a uma 'Nota Fiscal'.

Como se optou pela independência do modelo das PROPRIEDADES, foi criada uma classe chamada MODELO, que possui a mesma função da classe TIPOOBJETO do Modelo de Dados Universal. A diferença principal está no fato de que, sendo os modelos independentes, pode-se associar mais de um MODELO a um RECURSO ou DOCUMENTO ou ainda pode-se associar um MODELO a vários TIPOS DE RECURSOS diferentes. Como exemplo, é possível se ter os objetos 'Cliente' (TIPORECURSO) e 'Fornecedor' (TIPORECURSO) associados ao objeto 'PessoaJurídica' (MODELO), fazendo com que as PROPRIEDADES de um 'Cliente' (TIPORECURSO), tais como 'CGC', 'Endereço', 'Contato' entre outras, sejam as mesmas que as de um 'Fornecedor' (TIPORECURSO).

No Padrão Universal de Dados tem-se que a classe TIPOOBJETO, está relacionada com a classe ATRIBUTO num relacionamento de n-para-n da mesma forma que a classe MODELO se associa com a classe PROPRIEDADE através da classe PROPRIEDADEMODELO. A classe PROPRIEDADE MODELO determina quais são as PROPRIEDADES que podem ser associadas a um determinado MODELO e qual será o tipo do dado da PROPRIEDADE. Os tipos válidos são:

Número: armazena somente valores numéricos e a sua unidade de medida padrão, que pode ser uma das definidas na classe UNIDADEVÁLIDA.

Texto: armazena qualquer texto, sendo que o mesmo pode ser um texto livre, ou pode ter textos padrões associados, para facilitar a digitação.

Data: armazena valores na forma de data e hora.

Imagem: armazena informações no formato binário, normalmente imagens.

Domínio: armazena um relacionamento com a classe RECURSO sendo os filhos deste os valores possíveis do domínio.

Na FIGURA 36 tem-se as classes NÚMEROVÁLIDO, TEXTOVÁLIDO, DATAVÁLIDA, IMAGEMVÁLIDA e DOMÍNOVÁLIDO, como agregações da classe PROPRIEDADEMODELO. Optou-se pela estrutura de agregação e não por uma generalização, pelo fato de que esta última restringiria os objetos da classe PROPRIEDADEMODELO a um único tipo, ou seja, se for necessário armazenar o ‘código do plano de saúde’ e o ‘número da carteirinha do plano’ do ‘João’ necessita-se de duas PROPRIEDADES, uma para a ‘carteirinha’ e outra para o ‘plano’. Com o uso da agregação os dois dados podem ser armazenadas em conjunto sem a necessidade de duas PROPRIEDADES.

Os atributos **ValorPadrão**, **Fórmula**, **Nulo** e **Formato**, que estão presentes em algumas classes da agregação, são responsáveis por armazenar respectivamente o valor padrão que será atribuído a propriedade quando nenhum outro valor for informado, a fórmula que origina o valor da propriedade calculada, se o valor pode ser nulo ou não e o formato de exibição do mesmo.

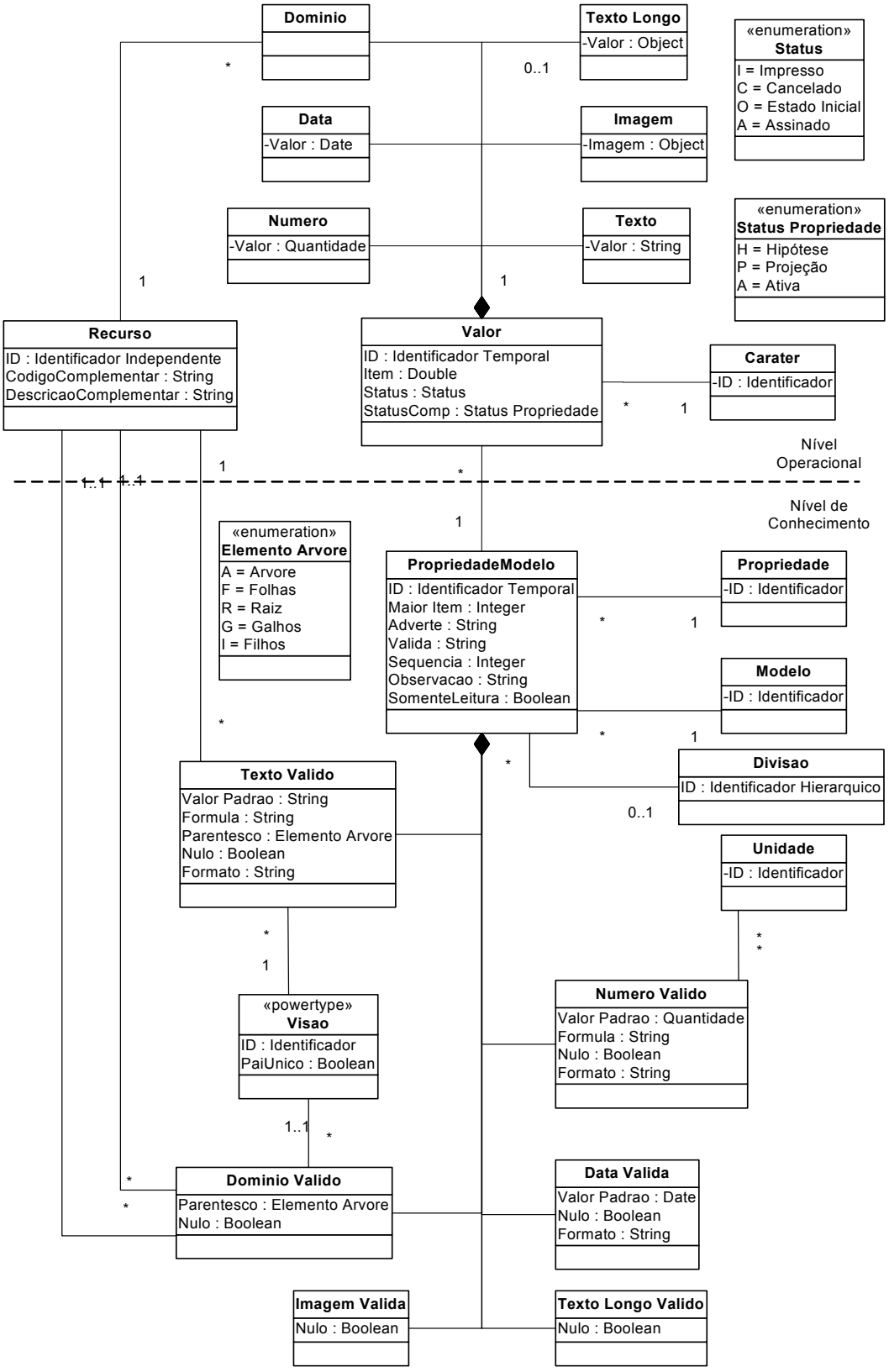

FIGURA 36 - Modelo do Caráter

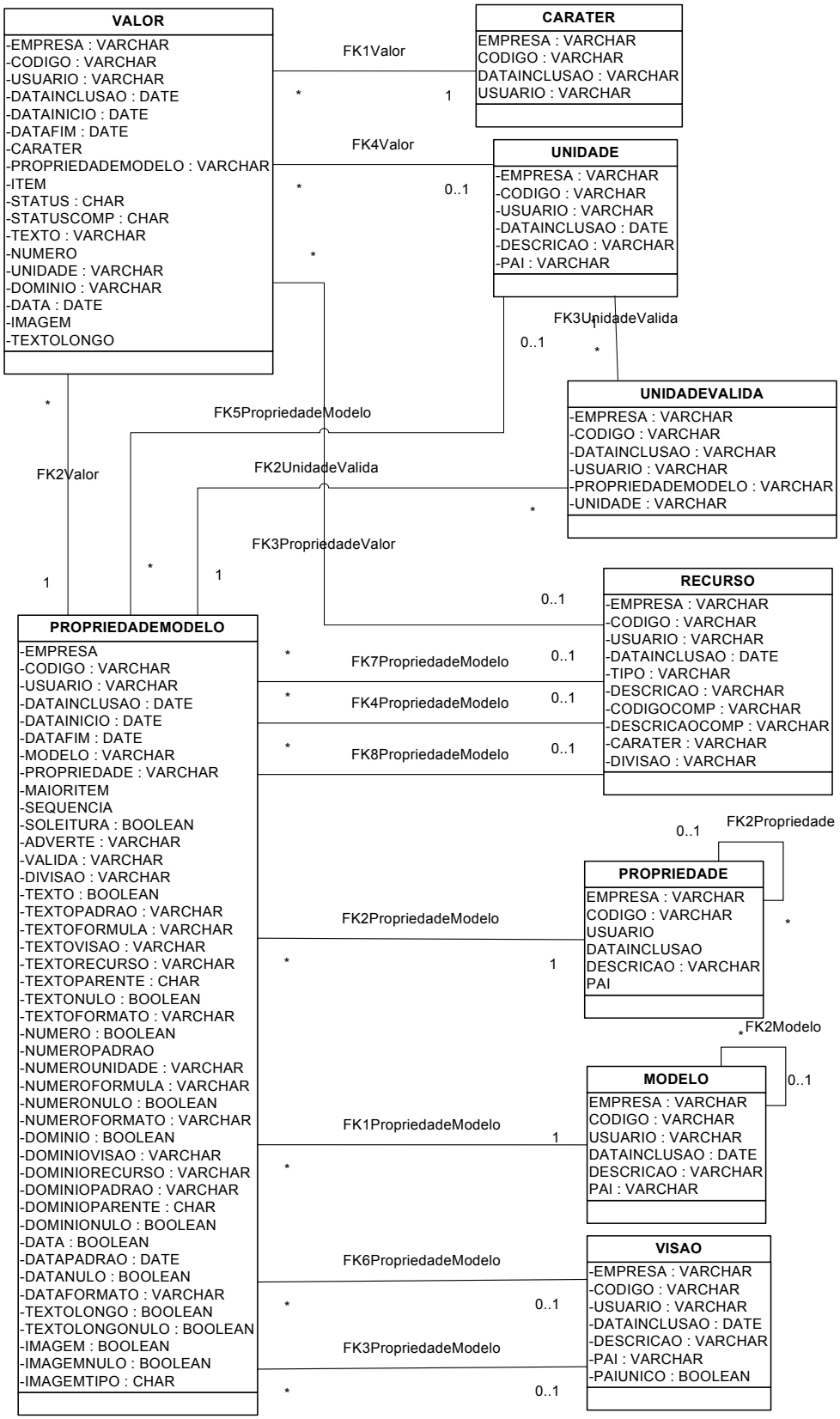

FIGURA 37 - Modelo Persistente do Caráter

A classe PROPRIEDADEMODELO contém ainda dois atributos destinados à consistência dos dados, um destinado à advertência, o **Adverte** e outro destinado à validação, o **Valida**. Ambos os atributos, como também o **Fórmula** possuem expressões que deverão ser interpretadas em tempo real e retornarão o valor resultante que, no caso do **Adverte** e **Valida**, serão textos contendo a mensagem a ser exibida para o usuário, ou nulo no caso de não haver erros.

O atributo **MaiorItem** determina a quantidade máxima de valores associados a uma mesma PROPRIEDADE. Exemplificando, se o **MaiorItem** tem valor igual a '1' (um) significa que a PROPRIEDADE em questão só poderá ter um valor associado a mesma em um ponto no tempo, já se o valor do mesmo é '5' (cinco), pode-se ter até cinco valores associados à PROPRIEDADE simultaneamente, ou seja, poder-se-ia ter no caso do exemplo anterior, do plano de saúde, até cinco planos de saúdes diferentes associados a mesma pessoa.

O atributo **Seqüência** determina a ordem na qual as PROPRIEDADES devem ser exibidas, o **SomenteLeitura** determina que o valor da PROPRIEDADE não poderá ser alterado pelo usuário e a **Observação** armazena um texto livre que objetiva auxiliar o usuário quando da digitação do valor da PROPRIEDADE.

Ao ser criado o Modelo Persistente do Caráter (FIGURA 37), optou-se por reunir as diversas classes agregadas à classe PROPRIEDADEMODELO na mesma. Esta alteração foi motivada pela simplificação do modelo e o aumento da performance no modelo relacional. Esta mesma solução foi adotada na classe VALOR e suas classes agregadas. O motivo desta decisão foi a simplificação do modelo, a melhora da performance e o fato de que as classes agregadas possuem um conjunto pequeno de atributos.

4.5.3.1 Interligação entre o modelo Caráter e os demais modelos

No tópico anterior foi visto que o modelo Caráter tem a classe MODELO que é composta por várias PROPRIEDADES, mas não foi visto como associar o valor das PROPRIEDADES a um RECURSO, e nem como associar um MODELO a um determinado RECURSO, que é o que será visto agora.

Como pode-se ver na FIGURA 38, todo RECURSO possui um CARÁTER, este último, por sua vez, é composto por vários VALORES, (FIGURA 36), que estão relacionado com as classes PROPRIEDADE e PROPRIEDADEMODELO este último se relacionando com MODELO que por sua vez está associados ao RECURSO através da classe MODELORECURSO (FIGURA 38), o MODELO pode ser associado ao TIPO RECURSO, forma mais comum, mas também pode ser associado a um RECURSO específico caracterizando-o individualmente. Na FIGURA 39 podemos visualizar o modelo persistente da classe MODELORECURSO.

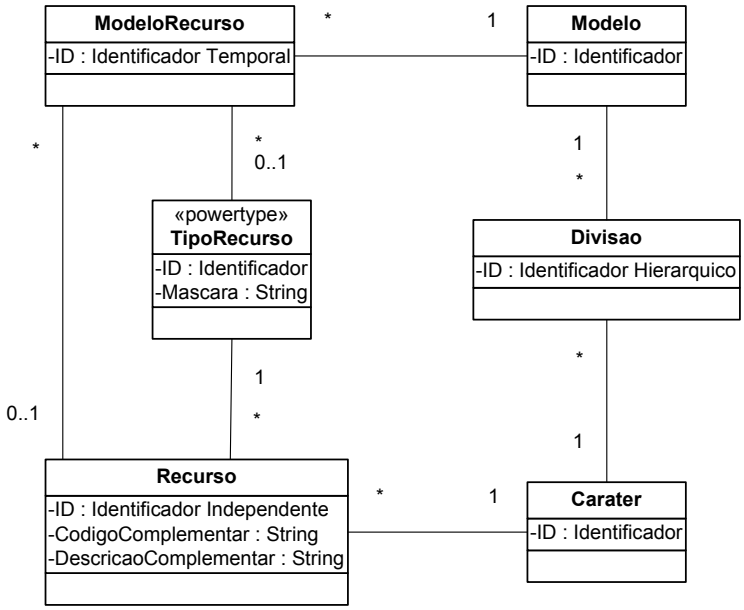

FIGURA 38 - Relacionamento entre o Modelo Recursos e Caráter

MODELORECURSO
«PK;FK1,2,3,4» -EMPRESA : VARCHAR
«PK» -CODIGO : VARCHAR
«FK1» -USUARIO : NUMBER
-DATAINCLUSAO : DATE
-DATAINICIO : DATE
-DATAFIM : DATE
«FK2» -MODELO : VARCHAR
«FK3» -TIPORECURSO : VARCHAR
«FK4» -RECURSO : VARCHAR

FIGURA 39 - Classe persistente MODELORECURSO

4.5.3.2 Controle de acesso ao Caráter

Da mesma forma como se restringe ou não o acesso a determinados RECURSOS a DIVISÕES específicas, pode-se restringir o acesso a classe VALOR

através da classe PROPRIEDADEMODELO (FIGURA 36). A classe PROPRIEDADEMODELO possui um relacionamento com a classe DIVISÃO que determina se a PROPRIEDADE em questão pertence a uma DIVISÃO específica, a todas as DIVISÕES (quando PROPRIEDADEMODELO não se relaciona a nenhuma DIVISÃO), ou ainda se a PROPRIEDADE possui valores diferentes para cada DIVISÃO.

Em relação ao controle de acesso de determinadas PROPRIEDADES por parte do USUÁRIO pode-se restringir o acesso às mesmas restringindo o TIPODOUSUÁRIO de acessar determinado MODELO através da classe DIREITOMODELO (FIGURA 40), que funciona de forma análoga a classe DIREITO RECURSO (tópico 4.4.2.1). Assim sendo, podemos ter o TIPORECURSO 'Funcionário' associado a dois MODELOS, um para a área industrial e outro para a área de Recursos Humanos, como não é aconselhável que os usuários da área industrial acessem informações sobre salários de seus colegas restringe-se o acesso ao MODELO dos Recursos Humanos liberando-se apenas o MODELO da área industrial. A FIGURA 41 apresenta a representação persistente da classe DIREITOMODELO.

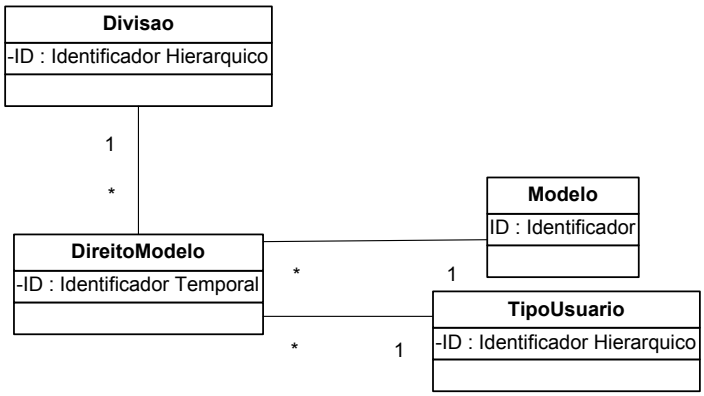

FIGURA 40 - Controle de acesso ao MODELO

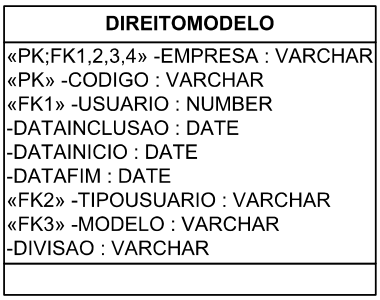

FIGURA 41 - Classe persistente DIREITOMODELO

4.6 Documentos

4.6.1 Padrão Contábil

> "A contabilidade pode ser vista, em termos gerais, como uma forma sistemática de se escrever a história econômica de uma organização. Sua finalidade é prover informações que possam servir de base para a tomada de decisões" (HAY, 1996 apud GORDON; SHILLINGLAW).

O modelo contábil não significa necessariamente o modelo clássico com plano de contas, contas de débito e contas de crédito. O mesmo pode ser qualquer sistema que responda a eventos de negócio que tenham conseqüências financeiras, não significando com isto que o mesmo seja aplicado somente a valores monetários. Este modelo pode ser aplicado utilizando-se horas trabalhadas ou Kw/h como unidade ao invés de valores monetários (FOWLER, 2000c).

É importante, na contabilidade, não somente mostrar o valor corrente de alguma conta, mas também poder detalhar cada mudança que afeta este valor. Em uma conta bancária, por exemplo, necessita-se registrar todos os saques e depósitos e não somente o saldo da conta; em um estoque, necessita-se registrar cada item adicionado e retirado do mesmo e não somente o saldo atual (FOWLER, 1997). Uma conta contábil é similar a um atributo de quantidade com um registro para cada mudança no seu valor, sendo que o balanço representa o valor atual da conta, que é o resultado da soma de todos os lançamentos ligados à mesma (FOWLER, 1997).

O padrão Contábil, mostrado na FIGURA 42 apresenta outra forma de se pensar sobre uma CONTA (do original *Account*). A mesma é um contenedor (do

original *container)* de LANÇAMENTOS (do original *Entry*) sempre que se cria um objeto do tipo LANÇAMENTO se associa o mesmo a uma CONTA. Além disto, a CONTA serve também para apresentar informações resumidas em um balanço, por exemplo (FOWLER, 2000c).

FIGURA 42- Padrão Contábil (*Account*; FOWLER, 2000c)

Normalmente, quando se pensa em CONTA, a mesma é associada a valores monetários, mas uma CONTA pode ser utilizada para se contabilizar valores de outra natureza, além dos valores financeiros. Por exemplo, ao se colocar e tirar da geladeira garrafas de vinho, pode-se pensar que são depósitos e saques (FOWLER, 2000c).

Um fato a ser levado em consideração no modelo contábil é que normalmente se possui uma grande quantidade de LANÇAMENTOS, o que torna necessária a otimização dos cálculos do balanço. A otimização depende do tipo de informação necessária, normalmente o saldo atual e os depósitos e saques de um dia ou período. Uma solução é colocar o saldo em um campo e utilizar os lançamentos para calcular os saldos anteriores (FOWLER, 2000c).

Nos LANÇAMENTOS, é importante que se tenha dois registros de data, um que informa a data do documento e outro a data do lançamento. Este procedimento se faz necessário para uma melhor análise do histórico dos acontecimentos. Para esta finalidade pode-se utilizar o tipo **REGISTRO DO TEMPO** visto anteriormente (FOWLER, 1997).

Uma das idéias principais da contabilidade, os lançamentos de duas entradas, no qual todo saque deve ser balanceado com um depósito, foi criada por um monge há milhares de anos. Para um contador, o dinheiro não é criado, ele apenas se movimenta de um lugar para outro (FOWLER, 2000c). É como se existissem vários potes de dinheiro e bens e fosse registrado o movimento entre os potes (FOWLER, 1997).

Esta idéia básica pode ser modelada de duas formas, uma delas é com uma única transação, a TRANSAÇÃO CONTABIL, relacionada a duas CONTAS, uma de débito e outra de crédito, sem a necessidade de composição de vários

LANÇAMENTOS (FIGURA 43). A outra forma é compondo uma TRANSAÇÃO CONTABIL com vários LANÇAMENTOS os quais totalizam zero quando somando seus débitos e créditos (FIGURA 44) (FOWLER, 2000c).

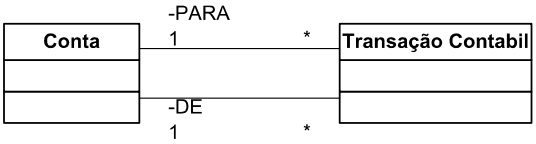

FIGURA 43 – Padrão Contábil com conta de débito e crédito (*Accounting Transaction without entries* ; FOWLER, 2000c).

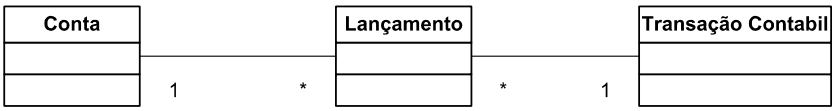

FIGURA 44- Padrão contábil com transação (*Accounting Transaction* ; FOWLER, 2000c)

Os lançamentos de duas entradas foram criados para facilitar a auditoria e prevenir que o dinheiro desapareça, ou seja, objetiva combater a fraude. Quando todos os lançamentos são feitos por computador, os registros das transações e a rastreabilidade das mesmas preenchem as necessidades de se encontrar fraudes, não sendo necessária a utilização dos lançamentos de duas entradas (FOWLER, 2000c).

Em um sistema contábil, as contas geralmente são compostas por grupos de CONTAS que são organizadas hierarquicamente. Por exemplo, pode-se ter as CONTAS de ‘medicina do trabalho’ e ‘alimentação’ dentro de ‘despesas com pessoal’, como também se pode ter as CONTAS de ‘restaurantes’ e ‘aluguéis de carros’ dentro da conta de ‘despesas com viagens’. Este tipo de estrutura pode ser suportada por uma hierarquia simples com CONTAS de detalhe e resumo (analítica e sintética) como mostrado na FIGURA 45. Uma das restrições que este modelo impõe é que só são permitidos LANÇAMENTOS associados às CONTAS de DETALHE e não às de RESUMO, as quais serão compostas pelos LANÇAMENTOS das contas subordinadas à mesma (FOWLER, 1997).

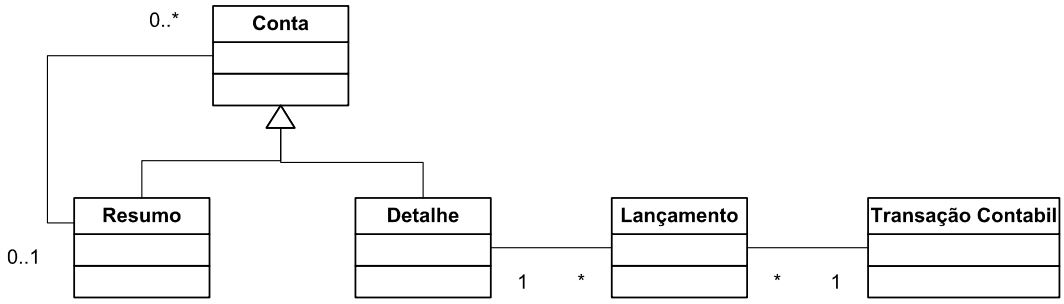

FIGURA 45- Modelo contábil com conta de resumo e detalhe (FOWLER, 1997)

4.6.2 Modelo Documentos

Este modelo foi o que sofreu mais alterações em relação à sua origem. Apesar deste modelo possuir sua origem no padrão contábil com transação (FIGURA 44), o mesmo foi estendido para modelar os documentos estruturados que servem como base à administração das empresas, tais como requisições e devoluções de estoque, notas fiscais, cartas de produção, cheques, ordens de compras, exames, entre outros.

A estrutura base do modelo Documentos é mostrada na FIGURA 46, este modelo se baseia no padrão Contábil com Transação (FIGURA 44) sendo que as classes CONTA, LANÇAMENTO e TRANSAÇÃO CONTÁBIL passaram a se chamar respectivamente de RECURSOS, ITENS e DOCUMENTO.

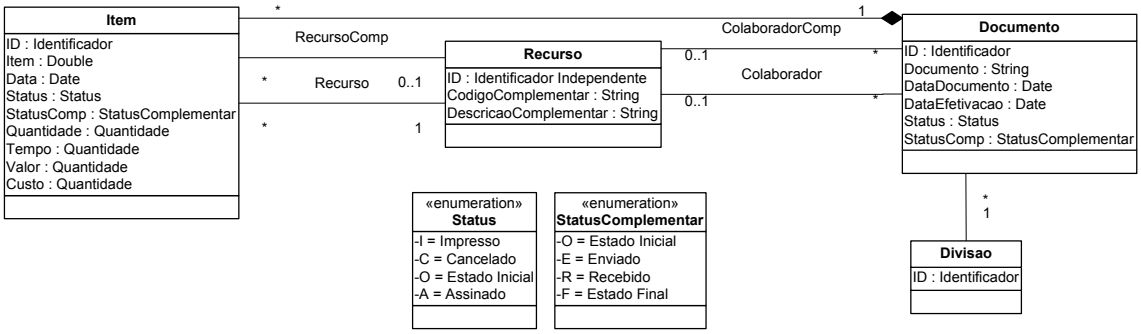

FIGURA 46 - Modelo Básico dos Documentos

A classe CONTA foi substituída pela classe RECURSO herdando da mesma a sua estrutura hierárquica flexível e transformando as CONTAS CONTÁBEIS em RECURSOS. Assim, possibilitando-se ter um único objeto que se refere a ‘materiais’, ‘clientes’, ‘contas bancárias’, ‘funcionários’ entre outros recursos, que são utilizados pelas diversas áreas da empresa.

Uma das diferenças existentes entre o modelo dos Documentos e o Padrão Contábil com Transação é que, enquanto no padrão contábil cada LANÇAMENTO possui uma CONTA associada, no modelo Documentos tem-se dois RECURSOS associados à classe ITEM. Na verdade os dois RECURSOS juntos equivalem a uma CONTA contábil única, e não como CONTA de débito e crédito como poderia ser imaginado. Este artifício é utilizado para facilitar o cadastramento das CONTAS no sistema e torná-lo mais flexível. Assim, para lançar uma despesa de combustível no setor comercial, basta associar o ITEM aos RECURSOS 'Combustível' e 'Área Comercial' que são RECURSOS independentes. Em um plano de contas que suporta apenas um relacionamento entre a CONTA e o LANÇAMENTO, deve-se ter as CONTAS de despesas associadas a cada setor que as consome em uma única CONTA, necessitando-se assim criar a CONTA 'Despesas de Combustíveis da Área Comercial', 'Despesas de Combustíveis com a Fábrica' e assim por diante, para cada setor. Esta estratégia cria um plano de CONTAS muito extenso principalmente quando se necessita um nível de detalhamento bastante fino.

Outra diferença está no fato do sistema contábil tradicional ter sua base nos LANÇAMENTOS de duas entradas, estratégia esta não utilizada no modelo Documentos. Como o sistema possui uma estrutura de manutenção de histórico robusta, a qual garante a completa rastreabilidade das alterações efetuadas no sistema, optou-se por não utilizar esta estratégia que tem sua origem neste fim (FOWLER, 2000c).

Para exemplificar o uso das classes DOCUMENTOS, ITENS e RECURSOS, será utilizado um sistema de "Controle de Horas de Produção". Neste sistema os operadores, possuem uma "Ficha de Controle de Produção", na qual, ao final de cada operação efetuada, anotam os seguintes dados: o que foi feito (ordens de produção), quantas peças foram feitas, quem fez (operadores), com o que (máquinas utilizadas) e quando (horário inicial e final de cada operação). Estes dados são utilizados para monitorar a evolução das ordens de produção na fábrica, os tempos de cada operação servem de base para o sistema de custos da empresa.

Para modelar a ‘Ficha de controle de produção’ deve-se criar os RECURSOS ‘Máquinas’, ‘Funcionários’ e ‘Ordens de Produção’, cada qual com sua estrutura hierárquica (os ‘funcionários’ e ‘máquinas’ organizados conforme os ‘setores da fábrica’ e as ‘ordens de produção’ organizadas por ‘clientes’ e ‘conjunto de produtos’). Para cada ‘Ficha de controle de produção’ deve ser criado um DOCUMENTO com a quantidade de ITENS igual à quantidade de operadores e máquinas utilizados em cada operação, ou seja, considerando que se tenha um operador, utilizando uma máquina em uma operação única, é necessário que um ITEM se relacione com o ‘Operador’ e a ‘Ordem de Produção’ em questão e o outro com a ‘Máquina’ e a ‘Ordem de Produção’ em questão. Além destas informações o ITEM possuirá a data e hora do inicio da operação no atributo **Data** a quantidade de tempo decorrente para efetuar a operação no atributo **Tempo** (que é o resultado da subtração da hora inicial e final) a quantidade de peças produzidas no atributo **Quantidade**, podendo ter ainda o custo da operação e o valor de venda, nos atributos **Custo** e **Valor**, não sendo estes obrigatórios. Salienta-se que os relacionamentos do DOCUMENTOS com o RECURSO não são necessariamente utilizados, já o relacionamento com a DIVISÃO é obrigatório pois todo o DOCUMENTO pertence a uma e somente uma DIVISÃO.

Estendendo-se o uso deste modelo, pode-se implementar um sistema de controle de estoque que controle as Notas Fiscais (NF) de entrada e de saída como também as requisições e devoluções internas de uma empresa. Para tal, foi criada a classe TIPODOCUMENTO que classifica os vários tipos de documentos controlados pelo sistema (FIGURA 47), sendo o mesmo, neste exemplo: ‘NF Entrada’, ‘Requisição’, ‘Devolução’, ‘NF Venda’ e ‘Controle de Produção’. Além da classe TIPODOCUMENTO foi criada também a classe TIPOITEM que neste exemplo poderá ser ‘Entrada’, ‘Saída’ ou ‘Produção’. Assim, quando entrar material no estoque, a ‘NF de entrada’ será registrada na classe DOCUMENTO que possui os seguintes atributos: **Documento**, contém o número da NF; **DataDocumento**, contém a data que consta como emissão da NF e a **DataEfetivação**, contém a data do recebimento da mercadoria. Além destes atributos, a classe DOCUMENTO possui dois relacionamentos que serão utilizados, o primeiro com a classe DIVISÃO, que

indica a qual divisão se refere o documento, e o relacionamento **Colaborador**, com a classe RECURSO, que identifica o 'Fornecedor' a qual a NF se associa (o **ColaboradorComp** é utilizado em alguns DOCUMENTOS nos quais são necessários dois RECURSOS para identificar o cliente, em sistemas da área médica é associado ao mesmo o convênio do cliente, por exemplo). Em uma NF pode-se ter vários ITENS, estes são do tipo 'Entrada' e possuem dois relacionamentos com a classe RECURSO, o primeiro é utilizado para identificar o material que está entrando no estoque e o segundo identifica em qual estoque esta entrando a mercadoria. A quantidade física, valor e custo são armazenados nos respectivos atributos. A 'NF de venda' possui o mesmo comportamento da 'NF de entrada', bastando modificar-se o TIPODOCUMENTO que passa a ser 'NF Venda' o TIPOITEM que passa a ser 'Saída' e o relacionamento 'Colaborador' será com um RECURSO do tipo 'Cliente'. Para registrar a movimentação interna dos materiais o DOCUMENTO de 'requisição' deve ser composto por, no mínimo, dois ITENS, um do Tipo 'Saída' que retira o material do estoque e outro de 'Entrada' que adiciona o material na respectiva 'ordem de produção' ou 'setor' consumidor. O DOCUMENTO 'devolução' tem seu funcionamento semelhante ao da 'requisição' diferenciando-se nos casos em que há transformação do material, neste caso é feita a 'entrada' de um 'produto' no estoque e a 'saída' dos 'materiais' da 'ordem de produção'.

Ao se criar a classe TIPO DOCUMENTO e TIPO ITEM para suportar vários tipos de documentos, necessita-se criar estruturas que validem se o TIPO ITEM é compatível com o TIPO DOCUMENTO e se os relacionamentos das classes DOCUMENTO e ITEM com a classe RECURSO são válidos. Exemplificando, não se pode ter um DOCUMENTO do tipo 'NF entrada' associado a um ITEM do tipo 'saída', para tal verificação a classe VALIDA DOCUMENTO ITEM será responsável por definir quais tipos de itens podem ser associados aos tipos de documentos. O atributo **Obrigatório** identifica se o TIPO ITEM obrigatoriamente faz parte do TIPO DOCUMENTO ou não.

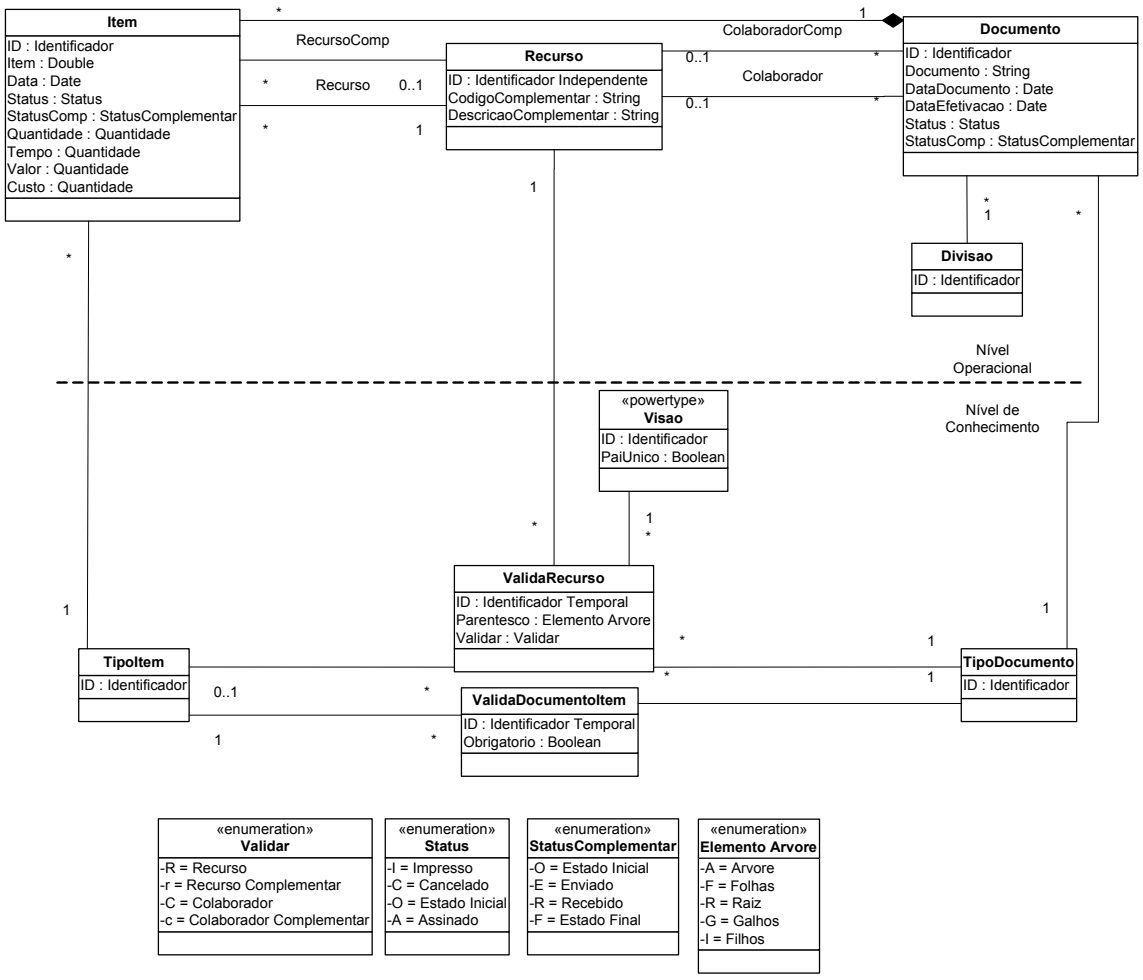

FIGURA 47 - Modelo Documentos com tipos

Outra verificação importante é não permitir que se venda ‘Funcionários’ ou que um RECURSO do tipo ‘Cliente’ seja requisitado para produção. A classe VALIDARECURSO é responsável por este tipo de validação, o relacionamento da mesma com a classe TIPODOCUMENTO é obrigatório e o atributo **Validar** identifica se o relacionamento com a classe RECURSO e VISÃO validarão o relacionamento **Colaborador, ColaboradorComp, Recurso** ou **RecursoComp**, podendo-se ainda ter relacionamentos específicos a determinados tipos de itens. Para exemplificar, suponha-se que na estrutura hierárquica dos RECURSOS, na VISÃO ‘principal‘ temos o RECURSO ‘Materiais em Estoque’ que é **pai** de todos os materiais em estoque e tem-se também o RECURSO ‘Consumidores’ e ‘Armazenadores’ que são “pais” das ‘ordens de produção’ e ‘estoques’, respectivamente. Assim poder-se-ia ter um

TIPODOCUMENTO ‘Requisição’ composto por dois TIPOITEM ‘saida’ e ‘entrada’ os quais devem se associar aos filhos do RECURSO ‘Materiais em Estoque’ no relacionamento **Recurso** e no relacionamento **RecursoComp** deveriam se associar aos filhos dos ‘Armazenadores’ e ‘Consumidores’ respectivamente. O fato de não especificar nenhum atributo **Validar** igual a ‘Colaborador’ e ‘ColaboradorComp’ especifica que os relacionamentos **Colaborador** *e* **ColaboradorComp** não são necessários neste TIPODOCUMENTO.

A necessidade crescente das empresas de se adequarem às normas de qualidade estabelecidas tornou necessário que sistemas de estoque fossem capazes de rastrear os materiais utilizados em um determinado produto, ou seja, ao ser informado o número de uma NF de venda, de um determinado produto, o sistema deve ser capaz de identificar as NF de entrada dos materiais utilizados na fabricação do mesmo.

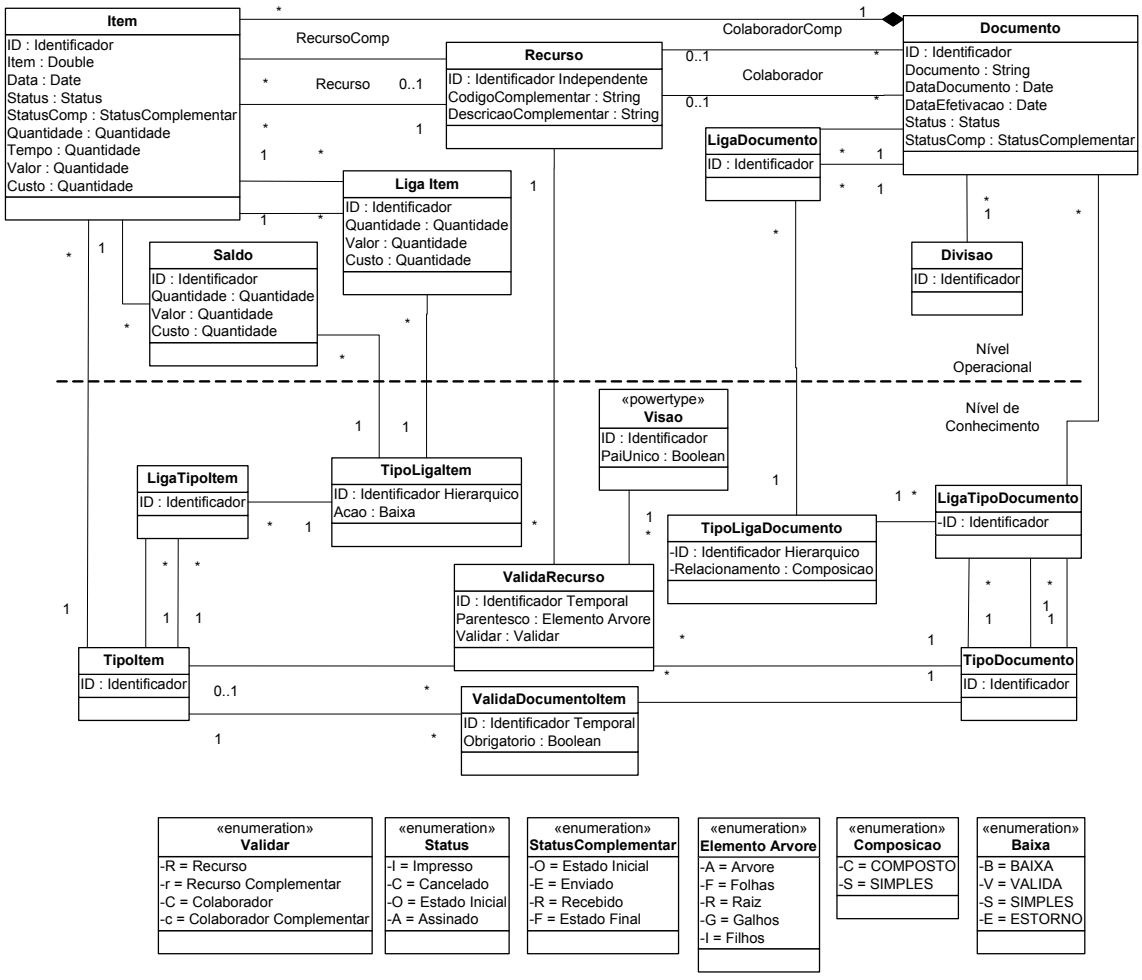

FIGURA 48 - Modelo Documentos

Para atender esta necessidade (entre outras) foram criadas classes que interligam DOCUMENTOS e ITENS entre si (FIGURA 48), possibilitando assim uma rastreabilidade completa dos documentos da empresa pela identificação da origem dos mesmos. As classes criadas foram: a classe LIGADOCUMENTO que interliga dois documentos; a classe TIPOLIGADOCUMENTO que identificará o tipo de interligação entre os documentos; a classe LIGATIPODOCUMENTO que determinará quais são os tipos de ligações válidas entre os documentos; a classe LIGA ITEM que interliga dois ITENS; a classe TIPOLIGAITEM que identificará o tipo de interligação entre os ITENS; e a classe LIGATIPOITEM que determinará quais são os tipos de ligações validas entre os ITENS. As estruturas de interligação dos DOCUMENTOS e ITENS são análogas.

Voltando ao exemplo do estoque, e supondo que é necessário ter a rastreabilidade do mesmo, se reconfiguraria o sistema da seguinte forma: a 'NF de entrada' continuaria igual, criar-se-ia um TIPOLIGAITEM e um TIPOLIGADOCUMENTO 'Rastreabilidade', os DOCUMENTOS do tipo 'Requisição', 'Devolução' e 'NF venda' não seriam mais compostos por dois objetos TIPOITEM um de 'entrada' e outro de 'saída', seriam compostos apenas pelo TIPO ITEM 'Entrada', na classe LIGATIPODOCUMENTO seria permitido que uma 'Requisição' tivesse sua origem em uma 'NF entrada' ou 'Devolução' uma 'Devolução' teria sua origem em uma 'Requisição' e a 'NF venda' teria sua origem na 'Devolução de Produção', na classe LIGATIPOITEM seria permitido que uma 'Entrada' fosse interligada a outra 'Entrada' com o atributo **Ação** igual a 'Valida'. Com esta configuração, ao ser lançada uma 'Requisição' deverá ser identificado a qual ITEM da 'NF de entrada' pertence o material que está sendo requisitado, criando-se assim um LIGAITEM com a quantidade requisitada, valor e custo proporcional à mesma, este lançamento substitui o lançamento de baixa que existia na configuração anterior e funciona da mesma forma para as 'Devoluções' e 'NF Vendas'. Por motivos de performance, foi criada a classe SALDO que armazena o saldo associado a um TIPOLIGAITEM de um determinado ITEM, a criação do SALDO se dá no momento da criação do ITEM (para os tipos de itens que são origens de outros tipos) com

quantidade, valor e custo igual ao do item que o mesmo se refere. A cada novo LIGAITEM o SALDO é baixado conforme a ação do TIPOLIGAITEM, sendo as mesmas as seguintes: 'Simples', não efetua baixa do saldo; 'Baixa', baixa saldo permitindo saldo negativo; 'Valida', baixa saldo se houver o mesmo; 'Estorno', estornará o saldo associado ao item a qual se refere o lançamento que está sendo estornado. Quando o SALDO de um determinado ITEM é zerado, o mesmo é destruído. Para determinar o SALDO total de um determinado 'produto:recurso', basta somar o SALDO dos ITENS do 'produto' e do 'estoque' em questão. Por ser uma classe redundante, que pode ser recriada caso necessário, não é necessário armazenar o histórico(*log*) da mesma.

Os inter-relacionamentos entre DOCUMENTOS e ITENS também são utilizados para interligar o 'pedido' com a 'NF venda' ou ainda a 'duplicata' com seu 'Pagamento', estes são exemplos onde a manutenção do saldo é desejada, já existem outros casos onde o relacionamento não envolve a baixa do saldo, que é o caso da 'Fatura' associada a 'Nota Fiscal'.

Para estender a configuração anterior, tornando possível controlar o contas a pagar, é necessário criar o TIPO DOCUMENTO 'Fatura', o TIPO ITEM 'Duplicata', o TIPOLIGADOCUMENTO 'Financeiro' com o atributo **Relacionamento** igual a 'composto' e o LIGATIPODOCUMENTO entre 'NF entrada' e 'Fatura'. Assim, ao informar os dados da 'NF Entrada', deverá ser informado, além dos dados já citados anteriormente, o valor das duplicatas associadas a NF com seus vencimentos, gerando assim um DOCUMENTO 'Fatura' que está relacionado com a respectiva NF através da classe LIGADOCUMENTO que possui o valor das duplicatas e seus vencimentos nos ITENS da 'Fatura'. Quando do pagamento da 'fatura', é criado um DOCUMENTO 'Cheque' que baixa a respectiva 'Duplicata' associada ao pagamento. A mesma configuração é utilizada em relação aos impostos associados às NF, os mesmos gerarão DOCUMENTOS e ITENS de crédito ou débito de impostos. Salienta-se que o atributo **Relacionamento** da classe TIPOLIGADOCUMENTO pode ser 'Composto', quando os documentos envolvidos possuem uma interligação muito

estreita, sendo vistos, às vezes, como um documento único, sendo nos demais casos 'Simples'.

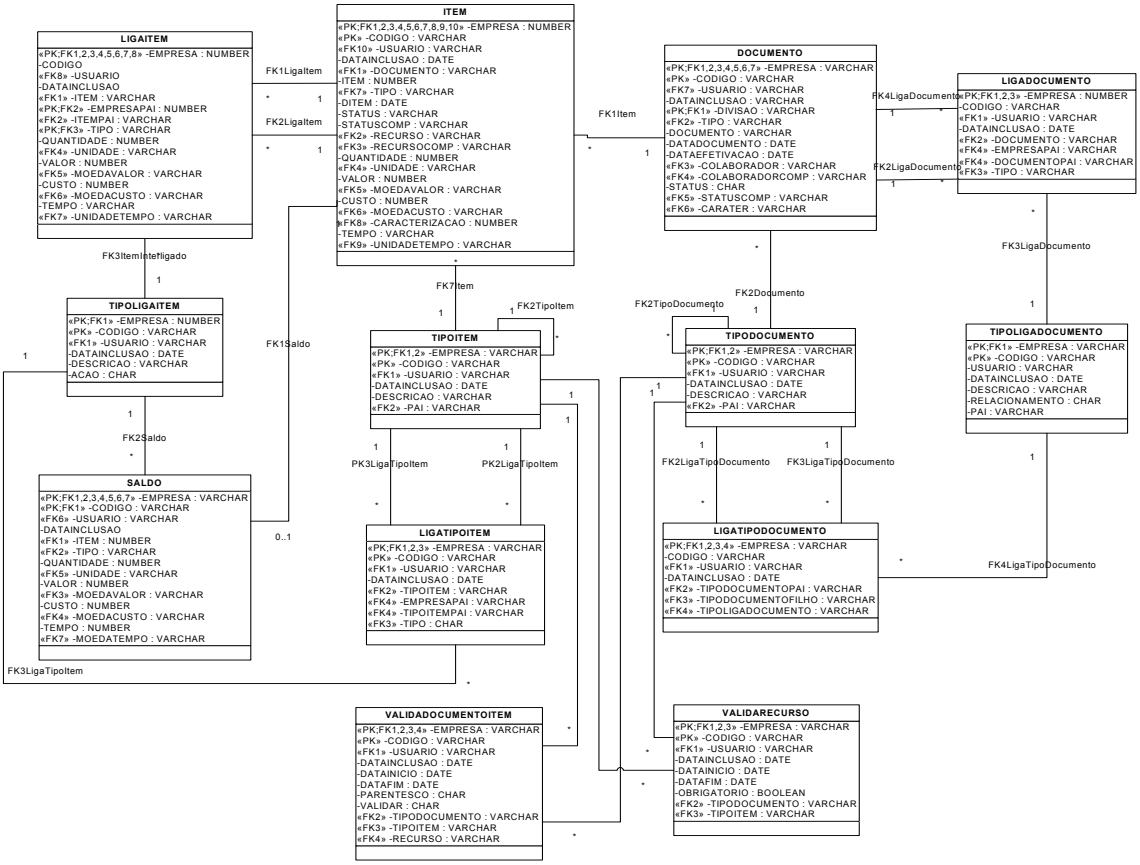

FIGURA 49 – Modelo persistente Documentos

A FIGURA 49 apresenta o modelo persistente para banco de dados relacional das classes do modelo Documentos.

Apesar do Modelo dos Documentos suprir as necessidades por informação de uma parcela de documentos, o mesmo ainda não consegue atender a documentos que contenham informações específicas ao documento ou ao item do documento. Para atender a estes casos, criou-se a classe MODELODOCUMENTO (FIGURA 50) que servirá para identificar qual MODELO (do Modelo Caráter) deve ser aplicado ao DOCUMENTO ou ITEM em questão. Relacionou-se as classes DOCUMENTO e ITEM com a classe CARÁTER que armazenará as propriedades específicas de cada DOCUMENTO. Com esta estrutura, pode-se armazenar o valor de qualquer

propriedade suportada pelo modelo Caráter. A FIGURA 51 apresenta o modelo persistente da classe MODELODOCUMENTO.

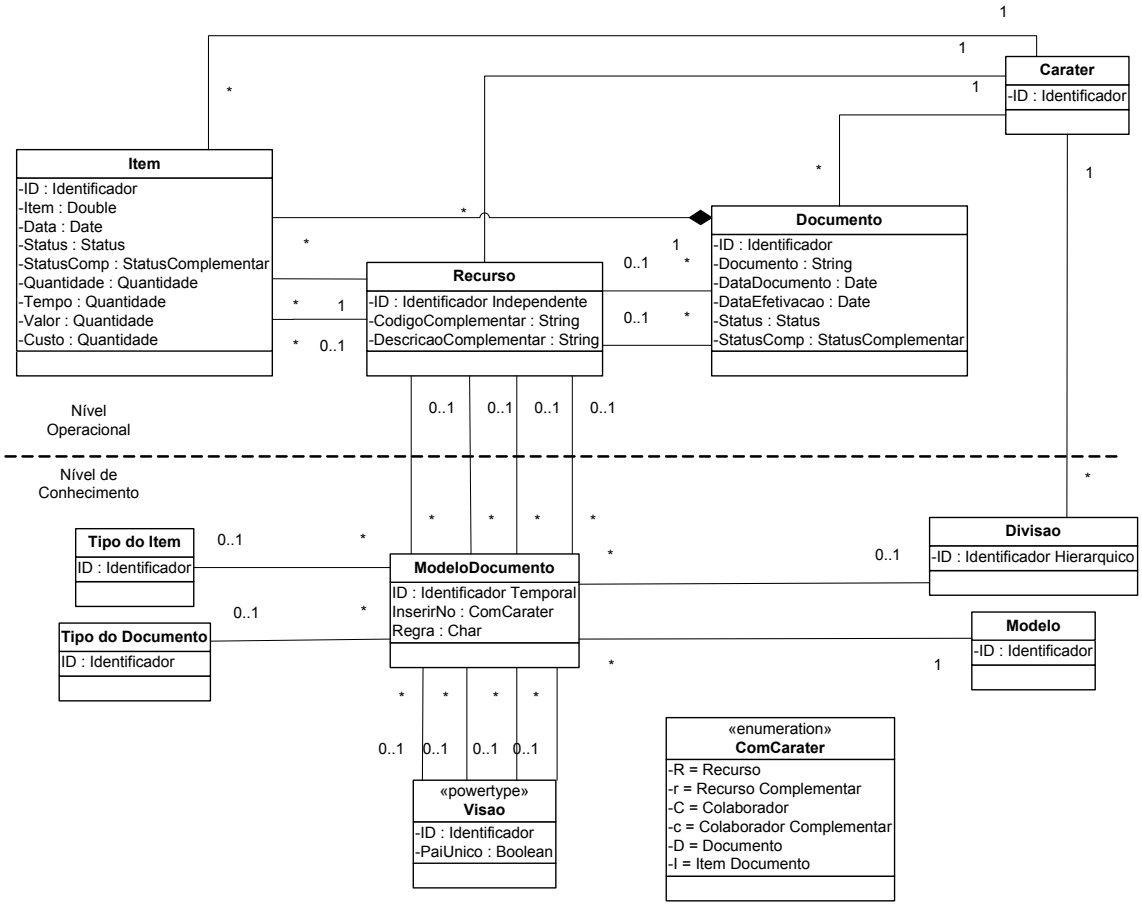

FIGURA 50 – Modelo ModeloDocumentos

MODELODOCUMENTO
«PK;FK1,2,3,4,5,6,7,8,9» -EMPRESA : VARCHAR
«PK» -CODIGO : VARCHAR
«FK1» -USUARIO : NUMBER
-DATAINCLUSAO : DATE
-DATAINICIO : DATE
-DATAFIM : DATE
«FK2» -MODELO : VARCHAR
«FK3» -TIPODOCUMENTO : VARCHAR
«FK4» -TIPOITEM : VARCHAR
«FK10» -VISAOCOLABORADOR : VARCHAR
«FK5» -COLABORADOR : VARCHAR
«FK11» -VISAOCOLABORADORCOMP : VARCHAR
«FK6» -COLABORADORCOMP : VARCHAR
«FK12» -VISAORECURSO : VARCHAR
«FK7» -RECURSO : VARCHAR
«FK13» -VISAORECURSOCOMP : VARCHAR
«FK8» -RECURSOCOMP : VARCHAR
«FK9» -DIVISAO : VARCHAR
-INSERENO : CHAR
-REGRA : VARCHAR

FIGURA 51 - Classe persistente ModeloDocumento

Seguindo o exemplo do sistema de estoque, suponha-se que seja necessário armazenar produtos perecíveis os quais tem-se que associar o seu lote de fabricação e a data de validade dos mesmos. Para atender a esta nova necessidade basta criar as PROPRIEDADES 'Lote de Fabricação' e 'Data de Vencimento', associar as mesmas ao MODELO 'Itens Perecíveis', através da classe PROPRIEDADEMODELO, especificando que as mesmas são do tipo 'String' e 'Data' respectivamente e associar o MODELO 'Itens Perecíveis' aos respectivos DOCUMENTOS. Para esta finalidade, utiliza-se a classe MODELODOCUMENTO na qual o modelo associado será o 'Itens Perecíveis', a DIVISÃO poderá ser nula habilitando este modelo para todas as DIVISÕES, associa-se o **Recurso** e **RecursoVisão** ao pai dos produtos que necessitam destas informações e configura-se o atributo **InserirNo** para inserir o modelo nos ITENS do DOCUMENTO 'NF de entrada'. Assim, ao se informar a 'NF de entrada' em todos os ITENS que estão associados ao modelo 'Itens Perecíveis', será necessário informar as propriedades 'Lote de Fabricação' e 'Data de vencimento'.

Nas NF também é necessário armazenar informações além das já citadas como 'Base de Cálculo do ICM', 'Alíquota de ICM' e 'Total da NF'. Estas informações serão consideradas PROPRIEDADES e poderão compor o MODELO "Informações sobre NF", que deve estar associado ao TIPODEDOCUMENTO 'NF de entrada' e deve associar o MODELO ao DOCUMENTO, assim, pode-se armazenar os diversos valores associados a uma 'NF' e também criar PROPRIEDADES que são resultantes de cálculos matemáticos definidos no atributo **Formula** da classe PROPRIEDADEMODELO, como é o caso da PROPRIEDADE 'Total NF'. Esta abordagem permite mudanças rápidas na forma de cálculo das 'NF' quando houver alteração por parte do governo sobre os cálculos, não se necessitando alterar os programas.

Outro problema que o modelo Documentos resolve está associados a materiais com características variáveis. Suponha-se que se fabrique cadeiras e que se tenha dez (10) modelos diferentes, sendo que estas podem ser forrados com cem (100) tipos de tecido diferentes, pintados em dez (10) cores diferentes com três (3) tipos de acabamentos diferenciados, assim, se fosse necessário cadastrar todos os produtos

resultantes, teríamos trinta mil (30000) produtos distintos e a cada novo acabamento criado ter-se-ia que cadastrar mais dez mil produtos (10000).

Para resolver este problema cria-se os dez produtos, referentes aos modelos, e associa-se a todos os DOCUMENTOS que se refiram a estes produtos o MODELO 'Características das Cadeiras', que é composto pelas PROPRIEDADES 'Cor', 'Tecido' e 'Tipo de Acabamento' que armazenam valores do tipo 'domínio'. Desta forma, todos os ITENS que se referem a estes produtos necessitarão que sejam informadas as suas PROPRIEDADES. Para se saber o saldo em estoque das 'Cadeiras A' de cor 'Branca', basta somar os SALDOS dos ITEM da 'Cadeira A' com a PROPRIEDADE 'Cor' igual ao valor 'Branca' .

Em um sistema laboratorial, pode-se ter centenas de exames diferentes com propriedades distintas em seus resultados, para esta situação deve-se criar um MODELO com as respectivas PROPRIEDADES para cada exame e associar o MODELO ao DOCUMENTO "Resultado do Exame" e ao exame específico na classe MODELODOCUMENTO, inserindo o modelo no ITEM do DOCUMENTO.

Outra situação que o modelo suporta é a necessidade de serem feitas perguntas específicas aos clientes quando os mesmos serão submetidos a um determinado exame. Perguntas do tipo peso, se é diabético ou não, se é fumante, entre outras. Salienta-se que apesar destas informações se relacionarem diretamente com o RECURSO 'Cliente' as mesmas só serão solicitadas quando o cliente solicitar determinado exame e não quando do seu cadastro no sistema. Para resolver esta situação, cria-se o modelo 'Perguntas para tomografia' e associa-se a mesma ao documento 'Requisição de Exame' e a família dos produtos da 'Tomografia' e se determina que o modelo deve ser relacionado com o 'Colaborador' ao invés de ser associado ao 'Documento' ou ao 'Item', como mostrado nos casos anteriores.

A união das características descritas no modelo Documentos torna o mesmo muito flexível, adequando-se às mais variadas necessidades informacionais associadas aos documentos empresariais estruturados.

O controle de acesso aos documentos é feito através da classe DIREITOTIPO DOCUMENTO (FIGURA 52), restringindo o acesso por parte do TIPOUSUÁRIO a

determinados TIPO DOCUMENTOS. Impede-se assim que um usuário, que não é do departamento pessoal, possa acessar o 'Envelope de pagamento da Folha' de seu colega de trabalho. A FIGURA 53 apresenta o modelo persistente da classe DIREITOTIPODOCUMENTO.

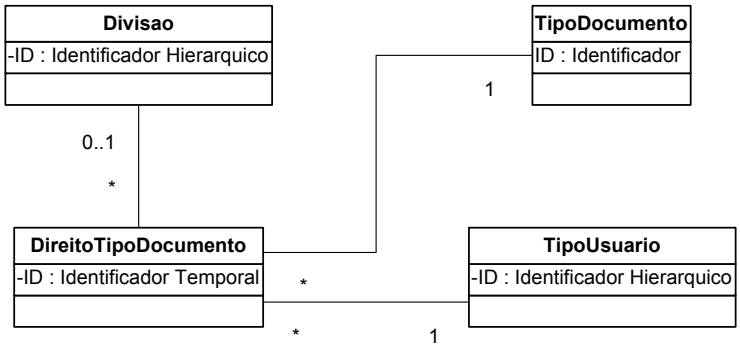

FIGURA 52 - Controle de acesso aos Documentos

DIREITOTIPODOCUMENTO
«PK;FK1,2,3,4» -EMPRESA : VARCHAR
«PK» -CODIGO : VARCHAR
«FK1» -USUARIO : NUMBER
-DATAINCLUSAO : DATE
-DATAINICIO : DATE
-DATAFIM : DATE
«FK2» -TIPOUSUARIO : VARCHAR
«FK3» -TIPODOCUMENTO : VARCHAR
«FK4» -DIVISAO : VARCHAR

FIGURA 53 - Classe Persistente DireitoTipoDocumento

4.7 Conclusão

A necessidade de criação de outras classes, além das que constam neste modelo, é um procedimento comum normalmente utilizado para modelar situações específicas e com complexidade não suportada pelo modelo base.

Os profissionais que irão desenvolver o sistema devem ser experientes, pois terão que utilizar freqüentemente técnicas de programação recursiva e fazer uso de componentes já existentes para assim obter-se a vantagem da utilização deste modelo.

Quando um novo profissional é adicionado à equipe de desenvolvimento, o tempo necessário para que o mesmo entenda o modelo existente e passe a utilizar os componentes já criados é maior do que quando se utiliza técnicas tradicionais de desenvolvimento de sistemas, normalmente já dominadas.

A deficiência do modelo relacional para lidar com estruturas hierárquicas é um dos problemas a serem enfrentados no desenvolvimento de sistemas que utilizem este modelo. CELKO (2000) apresenta várias técnicas úteis que auxiliam na solução deste problema. Ao desenvolver sistemas baseados neste modelo devem-se redobrar os cuidados com a performance do mesmo. As tabelas do nível operacional tendem a possuir muitas linhas fazendo-se necessário acessos otimizados às mesmas.

A utilização de modelos genéricos, como os vistos, gera um sistema flexível, com ciclo de vida mais longo e com necessidade menor de manutenções, mesmo quando da alteração das regras de negócio. Tais benefícios também requerem um pouco de sacrifício. Os problemas mais salientes associados a este tipo de modelagem são: a dificuldade de migração dos dados de um sistema anterior e as considerações em relação à performance. Tais considerações são importantes para que a equipe de desenvolvimento disponibilize mais tempo para as migrações e tenha um cuidado especial com os tempos de resposta do sistema. Outro fator importante para que um sistema com modelos genéricos dê certo é a necessidade de ter-se uma equipe experiente e capaz de realizá-lo (DORSEY; HUDICKA, 1999).

5 CONCLUSÃO

5.1 Implementações reais do modelo proposto

O modelo proposto foi aplicado em quatro empresas distintas: (i) indústria de autopeças, (ii) laboratório de análises clínicas, (iii) central de diagnósticos por imagens e, (iv) gerenciadora de medicina do trabalho.

Na indústria de autopeças o sistema implementado objetiva controlar o tempo de produção em todas as fases do processo produtivo. Para atingir este objetivo 'Produtos Prontos', 'Materiais', 'Máquinas', 'Funcionários' e 'Operações' se tornaram RECURSOS. As fichas do chão de fábrica se tornaram DOCUMENTOS que podem ter dois tipos de ITENS, um que armazena as 'Horas Máquina' e outro as 'Horas Homem', sendo que o **Recurso** do ITEM deve ser uma 'Máquina' ou 'Funcionário', o **RecursoComp** deve ser a 'Operação' efetuada ou pode ser uma 'Parada', sendo que a quantidade de peças produzidas é armazenada no atributo **Quantidade**, o tempo decorrido para efetuar a 'operação' é armazenado no atributo **Tempo** e a data e hora do início da operação é armazenado no atributo **Data.** Este sistema foi desenvolvido e implantado em três meses e está sendo utilizado pela área industrial como também pela área administrativa da empresa para detectar os problemas do processo produtivo e como fonte de informações para os sistemas de custos, planejamento industrial (PCP) e controle de qualidade.

No sistema do laboratório de análises clínicas e na central de diagnósticos por imagens o sistema objetiva administrar todo o processo produtivo desde a recepção do cliente emissão, dos resultados dos exames e faturamento dos mesmos contra os convênios. Os RECURSOS são 'Exames', 'Clientes', 'Médicos', 'Medicamentos', 'Tabelas de Preço', 'Convênios' e 'Planos' quando da recepção de um 'Cliente' é gerado um DOCUMENTO de 'Requisição' o qual é associado ao 'Cliente' e ao 'Convênio' do mesmo os ITEMS do DOCUMENTO armazenam os 'Exames' solicitados, os 'materiais' e 'medicamentos' com seus respectivos valores e o CARATER do ITEM conterá o resultado dos exames, sendo utilizado o modelo apropriado a cada tipo de exame. Este sistema atualmente gerencia mais de novecentos

tipos diferentes de exames, se adequando aos mesmos de forma individualizada e facilitando o acréscimo de novos modelos de exames, como também se adapta às mais variadas formas de negociação de mais de trinta convênios diferentes que são atendidos pelo cliente.

O sistema de medicina do trabalho objetiva administrar os exames que os empregados das empresas necessitam fazer periodicamente. O sistema tem como RECURSOS os 'Clientes', 'Setores', 'Funções', 'Funcionários', 'Exames', 'Engenheiros', 'Médicos', 'Riscos' e 'Tabelas de Preços' sendo que os 'Clientes' são organizados de forma hierárquica sendo subdivididos em 'Setores', 'Funções' e 'Funcionários' sucessivamente. Os exames periódicos a serem executados podem estar associados ao 'Cliente', 'Setores', 'Funções' ou 'Funcionários', sendo que os níveis mais baixos herdam os exames dos níveis superiores. O DOCUMENTO de requisição de exame é disparado após um evento do tipo 'Admissão', 'Demissão', 'Exame Periódico' ou 'Troca de Função' sendo que estes DOCUMENTOS se associam ao 'Cliente' e a seu 'Funcionário' tendo nos ITENS a associação com os 'Exames' a serem efetuados e aos respectivos 'Prestadores de Serviço'. O sistema atualmente gerencia os exames de trinta mil funcionários e de mais de novecentos clientes que utilizam o serviço. Por ser um setor com forte concorrência, o sistema tem se mostrado eficaz ao se adaptar rapidamente às novas políticas empresariais que objetivam a sustentação do negócio.

Como foi demonstrado, superficialmente, o modelo possui uma capacidade de adaptação não encontrada em sistemas desenvolvidos de forma tradicional. Os casos de uso de cada sistema foram implementados de forma personalizada beneficiando-se da reutilização do código e adequando o sistema às necessidades específicas de cada cliente.

5.2 Resultados práticos do uso de padrões

Comprovou-se que o modelo se manteve sem alterações, apesar das constantes modificações das regras de negócios. Outras conclusões resultantes da implementação do modelo nestas empresas estão listadas abaixo:

- A criação dos cadastros por parte do cliente pode se tornar simultânea ao desenvolvimento de casos de uso específicos, facilitando o desenvolvimento pela reutilização de componentes;
- A redução de custos de desenvolvimento é resultado da diminuição do tempo de desenvolvimento e também pela redução do tempo necessário por parte do pessoal da empresa na elaboração dos modelos.
- Os padrões tornam possível o desenvolvimento de módulos aplicados a setores específicos da empresa possibilitando a evolução em etapas sem comprometer a integração das diversas necessidades da empresa.

A utilização dos modelos como documentos representativos do processo produtivo facilita a interação da equipe técnica envolvida com o desenvolvimento de sistemas com a equipe de negócio da empresa.

Ao se entregar módulos do sistema em prazos cada vez menores, se cria credibilidade para o sistema em desenvolvimento e um ambiente favorável ao aprendizado com as mudanças, melhorando a qualidade do sistema percebida pelo usuário.

O fator crítico para a implantação de sistemas baseados em padrões de análise são as pessoas. O principal fator para o sucesso do sistema vem a ser a necessidade por técnicos em informática capacitados para lidarem com o desenvolvimento de sistemas e que sejam capazes de integrar com os profissionais do negócio de forma harmônica.

5.3 Trabalhos Futuros

O modelo proposto se adapta melhor ao controle das atividades empresariais do que ao planejamento das mesmas. Esta área de planejamento pode ser melhor explorada em trabalhos futuros de padronização. Os modelos de planejamento empresarial desenvolvidos pela Ph.D. Informática são, em sua totalidade, específicos as necessidades do cliente. Mesmo na bibliografia especializada encontra-se soluções específicas para determinado problema e não para modelos genéricos.

Na Ph.D. Informática o desenvolvimento de um *framework* específico para ser utilizado em conjunto com o modelo proposto agilizaria ainda mais o processo de desenvolvimento de novos sistemas e daria continuidade ao trabalho de padronização de todas as fases do processo produtivo da empresa possibilitando criar um diferencial competitivo através da padronização.

REFERÊNCIAS BIBLIOGRÁFICAS

ALHIR, S.S **Understanding the Unified Modeling Language (UML).** Methods & Tools. Matining & Associates, april, 1999.

AMBLER, S.W. **Persistence Modeling in the UML** Software Development: Agosto, 1999.

AMBLER, S.W. **Drawing Clean UML Diagrams Clarity Begets Adoption.** Disponível em:<http://www-4.ibm.com/software/developer/library/tip-uml/index.html> Acesso em: 10 dez. 2000a.

AMBLER, S.W. **How to Draw UML Activity Diagrams.** Disponível em:<http://www-4.ibm.com/software/developer/library/tip-uml/index.html> Acesso em: 10 dez. 2000b.

BELLOWS, J. **Activity Diagrams and Operation Architecture** 2000 CASTEK / CBD.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I**. The Unified Modeling Language User Guide**. Reading, MA: Addison-Wesley, 1999.

BROWN, W.J.; MALVEAU, R.C.; McCORMICK III, H.W.; MOWBRAY, T.J. **Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis.** New York, NY: John Wiley & Sons, 1998.

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAND, P.; STAL, M. **A System of Patterns: Pattern - Oriented Software Architecture.** New York, NY: John Wiley & Sons, 1996.

CELKO, J. **Joe Celko's SQL for Smarties: Advanced SQL Programming.** San Francisco CA: Morgan Kaufmann Publishers, 2000.

COAD, P; NORTH, D; MAYFIELD, M. **Object Models: Strategies, Patterns, and Applications.** Englewood Cliffs, NJ: Prentice-Hall, 1995.

CRINNION, J. **Evolutionary Systems Development: A Pratical Guide to the Use of Prototypes Within a Structured Systems Methodology.** Plenum Pub Corp, 1992.

DATE, C.J. **Guia Para o Padrão SQL.** Rio de Janeiro, RJ: Editora Campus Ltda, 1989.

DORSEY, P.; HUDICKA, J.R. **Oracle8 Design Using UML Object Modeling.** Berkeley, CA: Mc Graw-Hill, 1999.

ERIKSSON, H.E.; PENKER, M. **Business modeling with UML: Business Patterns at Work**. New York, NY: John Wiley & Sons, 2000.

FAYAD, M.E.; SCHMIDT, D.C.; JOHNSON, R.E. **Building Application Frameworks: Object-Oriented Foundations of Framework Design.** New York, NY: Wiley Computer Publishing, 1999.

FOOTE, B. Introduction In: **Pattern Languages of Program Design 3.** Reading, MA: Addison-Wesley, 1998.

FOWLER, M. **Analysis Patterns: Reusable Object Models**. Reading, MA: Addison-Wesley, 1997.

FOWLER, M.; SCOTT, K. **UML Distilled Second Edition: A Brief Guide to the Standard Object Modeling Language**. Reading, MA: Addison-Wesley, 2000.

FOWLER, M. **Patterns for Things That Change With Time.** Disponível em: <http://www.martinfowler.com> Acesso em: 10 dez. 2000a.

FOWLER, M. Prefácio. In: **San Francisco Design Patterns – Blueprints for Business Software.** Reading, MA: Addison-Wesley, 2000b.

FOWLER, M. **Accounting Patterns.** Disponível em: <http://www.martinfowler.com> Último acesso em: 10 dez. 2000c.

FOWLER, M. **Party.** Disponível em: <http://www.martinfowler.com> Acesso em: 10 dez. 2000d.

FOWLER, M.; HIGHSMITH, J. **The Agile Manifesto.** Software Development: Agosto, 2001.

FURLAN, J.D. **Modelagem de Objetos Através da UML: Análise e Desenho Orientados a Objetos.** São Paulo, SP: Makron, 1998.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Reading, MA: Addison- Wesley, 1995.

HALLE, B. von. **Business Rules Applied: Building Better Systems Using the Business Rules Approach.** New York, NY: John Wiley & Sons, 2001.

HAROLD, E.R. **XML Bible.** New York, NY: Hungry Minds, 2001.

HAY, D.C. **Data Model Patterns: Conventions of Thought.** New York, NY: Dorset House, 1996.

HRUBY, P. **Structuring Specification of Business Systems with UML (With an Emphasis on Workflow Management Systems).** Disponível em :<http.navision.com/default.asp?url=services/methodology/default.asp> Acesso em: dez 2000.

JACOBSON, I**. Object-Oriented Software Engineering: A Use Case Driven Approach.** Reading, MA: Addison-Wesley, 1992.

JACOBSON, I**.;** GRISS, M.; JONSSON, P. **Software Reuse: Architecture Process and Organization for Business Success.** Reading, MA: Addison-Wesley, 1997.

JACOBSON, I**.;** BOOCH, G.; RUMBAUGH, J. **The Unified Software Development Process**. Reading, MA: Addison-Wesley, 1999.

JÉZÉQUEL, J.M.; TRAIN, M.; MINGINS, C. **Design Patterns and Contracts.** Reading, MA: Addison-Wesley, 2000.

KOBRYN, C. **UML 2001: A Standardization Odyssey.** Communication of The ACM. October, 1999/Vol 42, no. 10.

LINDA, R. **The Patterns Handbook: Techniques, Strategies, and Applications.** Cambridge: Cambridge University Press, 1998.

LARMAN, C. **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design.** Upper Saddle River, NJ: Prentice-Hall, 1998.

MARSHALL, C. **Enterprise Modeling with UML – Designing Successful Software through Business Analysis.** Reading, MA: Addison-Wesley, 2000.

MARTIN, R.C. **UML Tutorial: Collaboration Diagrams.** C++ Report: nov/dec, 1997.

MARTIN, R.C. **UML Tutorial: Finite State Machines.** C++ Report: jun, 1998a.

MARTIN, R.C.; **UML Tutorial: Use Case Diagrams.** C++ Report: nov/dec, 1998b

MULLER,R.J.; **Database Design for Smarties: Using UML for Data Modeling.** San Francisco CA: Morgan Kaufmann Publishers, 1999.

OGAWA, A.; **Da Lata ao Carro: Como se Faz um Automóvel na fábrica mais moderna do mundo**. Quatro Rodas: set, 2001.

OVERGAARD, G.; SELIC, B.; BOCK, C. **Object modeling with OMG UML - Tutorial Séries – Behavioral Modeling** Disponível em: <http://www.celigent.com/omg/umlrtf/tutorials.htm> acesso em: 10 dez 2000.

POPPENDIECK, M. **Lean Programming.** Software Development: may 2001

PREE, W. **Design Patterns for Object-Oriented Software Development.** Reading, MA: Addison- Wesley, 1995.

RATIONAL **The UML and Data Modeling.** Disponível em: <http://www.rational.com > acesso em: 10 dez 2000

RISING, L. **The Patterns Handbook: Techniques, Strategies, and Applications.** Melbourne: Austrália, 1998.

REED JR., P.R. **Desenvolvendo Aplicativos com Visual Basic e UML.** São Paulo, SP: Makron, 2000.

RUMBAUGH, J. Et.al. **Modelagem e Projetos Baseados em Objetos**. Rio de Janeiro, RJ: Editora Campus Ltda, 1994.

SUTHERLAND, J. **The Object Technology Architecture: Business Objects for Corporate Information Systems.** Disponível em :http://jeffsutherland.org/ , 1997

VLISSIDES, J. **Pattern Hatching: Design Patterns Applied.** Reading, MA: Addison- Wesley, 1998.

WOMACK, J.P.; JONES, D.T.; ROOS, D. **A Máquina que Mudou o Mundo.** Rio de Janeiro, RJ: Campus, 1997.