Universidade Federal da Paraíba

Centro de Ciências Tecnológicas

Departamento de Sistemas e Computação

Uma Ferramenta CASE para a Metodologia FADO

Francisco Augusto Andrade Maia

Campina Grande - PB

Fevereiro de 1999

Francisco Augusto Andrade Maia

Uma Ferramenta CASE para a Metodologia FADO

Dissertação submetida à Coordenação de Pós-Graduação em Informática do Centro de Ciências e Tecnologia da Universidade Federal da Paraíba como requisito parcial para a obtenção do grau de Mestre em Ciência (M. Sc).

Área de Concentração: Ciência da Computação

Linha de pesquisa: Sistemas de Informação e Banco de Dados

Orientador: Prof. Ulrich Schiel, Dr.

Campina Grande

Universidade Federal da Paraíba

M217f Maia, Francisco Augusto Andrade.
Uma ferramenta CASE para a metodologia FADO / Francisco Augusto Andrade Maia. Campina Grande, 1998.
128 f.

Inclui CD-ROM.
Dissertação (Mestrado em Informática) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1998.
Referências.
"Orientação : Prof. Dr. Ulrich Schiel".

1. Bases de Dados. 2. Sistemas de Informação. 3. Projetos de Sistemas - Metodologia. 4. Dissertação - Informática. I. Schiel, Ulrich. II. Universidade Federal da Paraiba - Campina Grande (PB). III. Título

CDU 004.65(043)

UMA FERRAMENTA CASE PARA A METODOLOGIA FADO

FRANCISCO AUGUSTO ANDRADE MAIA

DISSERTAÇÃO APROVADA EM 25.02.1999

PROF. ULRICH SCHIEL , Dr.
Orientador

PROFª MARIA DE FÁTIMA Q. V. TURNELL, Ph.D
Examinadora

PROF. ALBERTO HENRIQUE FRADE LAENDER, Ph.D
Examinador

CAMPINA GRANDE – PB

À minha querida e paciente esposa e filhos.

Dedicatória

À DEUS, pela força concedida.

À minha esposa e filhos pelo amor, apoio, compreensão e paciência.

Ao Prof. Dr. Ulrich Schiel, pela orientação, compreensão e amizade que sempre prevaleceu nas nossas discussões.

À todas as pessoas que de alguma maneira me incentivaram a concluir este trabalho.

Agradecimentos

Ao SEPROCE pela oportunidade dada e apoio oferecido.

À Universidade Federal da Paraíba pela oportunidade oferecida.

Aos professores do curso de Pós-Graduação em Informática.

Aos funcionários da Miniblio.

Aos funcionários do COPIN pela atenção e presteza dedicada: Ana Lúcia Guimarães e Vera Lúcia de Oliveira.

Ao pessoal do LABCOM: Alberto e Lilian.

Ao Dr. Ariosto Holanda, secretário da Ciência e Tecnologia do Ceará (SECITECE) pela confiança.

Ao Dr. Celestino, Diretor do Instituto do Software, pelo incentivo.

Aos amigos que deixei em Campina Grande, especialmente ao paraibano César Augusto, ao baiano Aídre e ao paraense Agnes.

À IBM pelo apoio financeiro concedido.

Aos meus amigos do SEPROCE pela FORÇA e incentivo.

À banca examinadora pelas sugestões apresentadas (Prof. Dr. Laender e especialmente a Profa. Dra. Fátima Turnell).

Resumo

O grande problema enfrentado pelos desenvolvedores de software sempre foi e ainda é a complexidade crescente dos *softwares* a serem desenvolvidos. Para minimizar este problema surgiram métodos de trabalho e ferramentas computadorizadas que orientam e ajudam o desenvolvedor nesta complexa tarefa que é construir *software*.

O nosso trabalho foi desenvolver uma ferramenta CASE *front end* baseada na metodologia FADO, que é uma metodologia orientada a objetos. A ferramenta foi projetada para ser flexível, fácil de usar, e abrigar uma consistência entre os diagramas. Flexível, para permitir a adição de outros métodos e acoplamento a outras ferramentas, em resumo, a sua atualização. Fácil de usar, para permitir uma disseminação e um prazer em usa-la. Consistência entre os diagramas, para facilitar a manutenção dos projetos construídos.

Finalmente, gerar os *scripts* de banco de dados, e imprimir os diagramas e formulários confeccionados na construção do projeto, sendo a base da sua documentação.

Abstract

The great problem faced by the software develops always went and it is still to growing complexity of the softwares they be developed. To minimize this problem they appeared work methods and computerized tools that guide and they help the develop in this complex task that is to build software.

Our work went develop a tool it CASE front end based on the methodology FADO, that is a methodology oriented to objects. The tool was projected to be flexible, easy to use, and to shelter a consistency among the diagrams. Flexible, to allow the addition of another methods and joining to other tools, in summary, its modernization. Easy to use, to allow a contamination and a pleasure in you use it. Consistency among the diagrams, to facilitate the maintenance of the built projects.

Finally, to generate the database scripts, and to print the diagrams and forms made in the construction of the project, being the base of its documentation.

Sumário

ÍNDICE DE FIGURAS

1. Introdução

Este capítulo é dedicado à apresentação das motivações que nos levaram ao desenvolvimento deste trabalho de dissertação, destacando os principais objetivos e apresentando a estrutura dos capítulos seguintes.

1.1 Motivação

O que nos motivou a realizar este trabalho, a construção de uma ferramenta CASE para a Metodologia FADO - CASE FADO, foi o problema que hoje enfrentamos no uso de métodos de desenvolvimento de Sistemas de Informação, principalmente com os métodos orientado a objetos, com a geração de um grande número de diagramas, formulários e tabelas, dificultando a atualização e a integração dos mesmos. Por exemplo, quando se atualiza um diagrama quase sempre é necessário alterar vários ou todos os formulários além dos diagramas. O uso desses métodos de forma manual se torna uma tarefa enfadonha e sem consistência, levando o projetista a seguir apenas alguns passos da metodologia escolhida, em vez de ser "guiado" por ela em todo o processo de desenvolvimento de um sistema de informação. Posto isto, fica evidente a necessidade de uma ferramenta computadorizada que auxilie o projetista a desenhar, consistir, documentar, refinar e manter o sistema projetado, minimizando a incidência de erros de projeto.

Voltando um pouco no tempo, podemos apresentar um outro problema de grande importância no desenvolvimento de sistemas que está relacionado com o que ficou conhecido como "crise do software" [Jones'90]. Este termo, surgido no final da década de 60, é caracterizado pela incapacidade de atender a uma demanda de sistemas cada vez mais complexos, com uma melhor qualidade a um custo menor e um tempo menor de desenvolvimento, causando um alto grau de insatisfação dos usuários, que não chegavam a utilizar ou receber grande parte do software encomendado. Uma das alternativas para a solução deste problema foi o desenvolvimento de métodos e técnicas, tais como os métodos estruturados, criados para sistematizar o processo de desenvolvimento de sistemas. A maior parte desses métodos tem utilizado diagramas e grafos para modelar sistemas, já que diagramas são intuitivos, rápidos de aprender e fáceis de usar [Missikoff'96]. A partir da existência desses métodos veio a necessidade de construir ferramentas para automatizá-los. A disciplina que deu origem a estas ferramentas é denominada de CASE (Computer Aided Software Engineering).

O surgimento de ferramentas CASE trouxe vantagens que favoreceram o

desenvolvimento de sistemas. Em [Fisher'90] são enumeradas algumas dessas vantagens, são elas:

- Especificações completas de requisitos: incentivar os desenvolvedores de software a especificar completamente os requisitos do sistema é objetivo das ferramentas CASE de análise e especificação de requisitos;
- Especificações detalhadas do projeto: as ferramentas CASE são ferramentas para o projeto e não para a documentação posterior;
- Especificações atuais do projeto: as ferramentas CASE oferecem dispositivos para manter as especificações do projeto atualizadas em face da facilidade de manutenção contínua dessas especificações;
- Redução do tempo de desenvolvimento: a especificação completa da arquitetura do software reduz, quando não elimina, a perda de tempo com códigos desnecessários.

As ferramentas CASE estão altamente relacionadas com os métodos que implementam. No entanto esses procedimentos existentes na engenharia de software desde do seu aparecimento na década de 70, através das metodologias estruturadas, estão em constante evolução. Desta forma, a ferramenta que automatiza um método também deve evoluir para não se tornar obsoleta. Deve-se por isso, projetar uma ferramenta de forma que ela permita uma fácil evolução.

Além disso, o que ocorre na maioria das vezes, é a passagem de métodos que antes eram utilizados de forma tradicional, utilizando-se lápis e papel, para outra maneira de trabalhar só que usando teclado, mouse e monitor, ou seja, não observando os aspectos tecnológicos que um sistema computadorizado pode oferecer. Para caracterizar uma ferramenta além de permitir o desenvolvimento de diagramas gráficos é necessário também utilizar recursos como verificação de regras, geração automática de componentes da especificação e armazenamento das informações do sistema.

Uma quantidade extensa de ferramentas apareceu no mercado para atender fases específicas do ciclo de vida do software, principalmente as fases de programação e codificação. Isto significa que o desenvolvimento tradicional de software dá ênfase maior às fases finais do ciclo de vida. Segundo [Kelly'92][Basili'84] erros que ocorreram durante a fase de especificação são de 10 a 100 vezes mais difíceis de consertar durante as fases finais do ciclo de vida do sistema do que durante a fase de análise e projeto. Em [Comp87] foram feitos estudos onde calcularam que 64% das falhas de software surgiam nas fases de análise e projeto e que se detectavam 30% destes erros antes que o software fosse encaminhado ao teste de aprovação. Isto reforça a necessidade de projetos desenvolvidos corretamente na primeira vez. Estes seriam

alguns dos motivos para estudar o aperfeiçoamento do processo de análise e projeto utilizando a tecnologia CASE.

Muitas ferramentas CASE que abordam fases de análise e projeto relacionadas com as metodologias existentes apareceram. No entanto, essas ferramentas não obtiveram o sucesso esperado. Alguns fatores contribuíram para as dificuldades encontradas nessas ferramentas CASE, tais como:

- as metodologias e técnicas adotadas são resultados de estudos recentes quando comparadas com outras áreas de engenharia;
- quando essas ferramentas apareceram havia uma grande expectativa, no entanto com pouca experiência e poucos recursos as ferramentas não supriam as necessidades dos desenvolvedores;
- os projetistas não desenvolviam os sistemas de forma disciplinada e sistemática, tendo pouco conhecimento dos métodos existentes;
- o custo das ferramentas era e ainda é bastante alto. As empresas comerciais, com o objetivo de atingir esta parcela do mercado, passaram a desenvolver diversas ferramentas CASE. No entanto faltava uma análise mais detalhada para o desenvolvimento dessas ferramentas;
- muitas vezes as ferramentas não são utilizadas no período de formação dos desenvolvedores de sistemas.
- a empresa quer investir na tecnologia CASE e não sabe quais os critérios para escolher uma ferramenta adequada ao seu trabalho. Em [SEI95] são apresentados alguns critérios para selecionar a tecnologia apropriada para a equipe de projetistas e a empresa.

1.2 Objetivos da Dissertação

O principal objetivo deste trabalho é descrever a ferramenta CASE que desenvolvemos para a metodologia FADO (**F**erramenta de **A**nálise e **D**esenvolvimento **O**rientada a Objetos) [Furtado93], explicando de uma forma detalhada como essa metodologia deve se comportar nesta nova situação. Essa metodologia esta inserida no contexto do ambiente de desenvolvimento e gerência de banco de dados DYNAMO (**DYNA**mic **M**anagement of **O**bjects) [Schiel'92].

Apesar das vantagens oferecidas pelas ferramentas CASE, ainda existem barreiras, como as apresentadas no seção 1.1 acima, para que elas sejam usadas. Com o objetivo de cobrir

deficiências, aperfeiçoar e facilitar o processo de análise e projeto, a ferramenta CASE/FADO foi elaborada destacando aspectos não vistos em uma mesma ferramenta de análise, dos quais podemos mencionar:

- permitir modelagem de características temporais;
- identificação e representação de tipos adicionais de relacionamento e abstração (como os relacionamentos dinâmico, agrupamento);
- representação do comportamento dinâmico do sistema como um todo (troca de mensagens relacionada ao controle de eventos e "*triggers*");
- garantia de consistência entre os diversos diagramas e formulários.
- possibilidade de gerenciamento dos projetos que estão sendo desenvolvidos, incorporando uma ferramenta de gerência de projetos;
- apresentação de uma estrutura extensível e flexível podendo-se facilmente adicionar novos métodos e estender o existente;
- após a adição de novos métodos, possibilidade de que um determinado sistema possa ser visualizado por diferentes modelos. Desta maneira, o usuário terá diferentes perspectivas de um mesmo sistema;

O processo de desenvolvimento da ferramenta CASE/FADO foi feito seguindo-se as características inerentes ao modelo TOM (Temporal Object Model) [Schiel'91], que permite a modelagem de banco de dados seguindo o paradigma da Orientação a Objetos (OO). A ferramenta CASE/FADO dá uma ênfase maior às etapas iniciais do ciclo de vida do sistema de informação. Serão utilizadas as informações pertencentes à fase de projeto para geração de esquemas de banco de dados, e auxilio a um gerador de código. A ferramenta foi projetada de maneira a permitir a incorporação de um módulo de gerência de projetos. No caso a ferramenta escolhida foi a GEPRO, objeto de uma outra dissertação de mestrado [Torres'96].

Com o intuito de dar uma visão do contexto da dissertação, apresentamos de forma gráfica o ambiente DYNAMO (Figura 1-1), que objetiva dar suporte ao desenvolvimento de aplicações mais avançadas de sistemas de processamento da informação e do conhecimento. Na referida figura podemos visualizar uma interface de manipulação, onde o usuário faz suas solicitações ao sistema e recebe os resultados. Essas solicitações e resultados são processados pelo Gerenciador de Objetos Temporais e Ativos (GOTA). O GOTA faz todas as conversões entre o ambiente de aplicação e o ambiente de gerência das informações e monitora os diversos acessos aos objetos. O monitor de eventos detecta a ocorrência de eventos pela análise de mensagens recebidas do gerenciador de objetos ou pela consulta ao relógio interno e solicita ao

gerenciador de objetos a execução das ações específicas de acordo com o esquema conceitual.

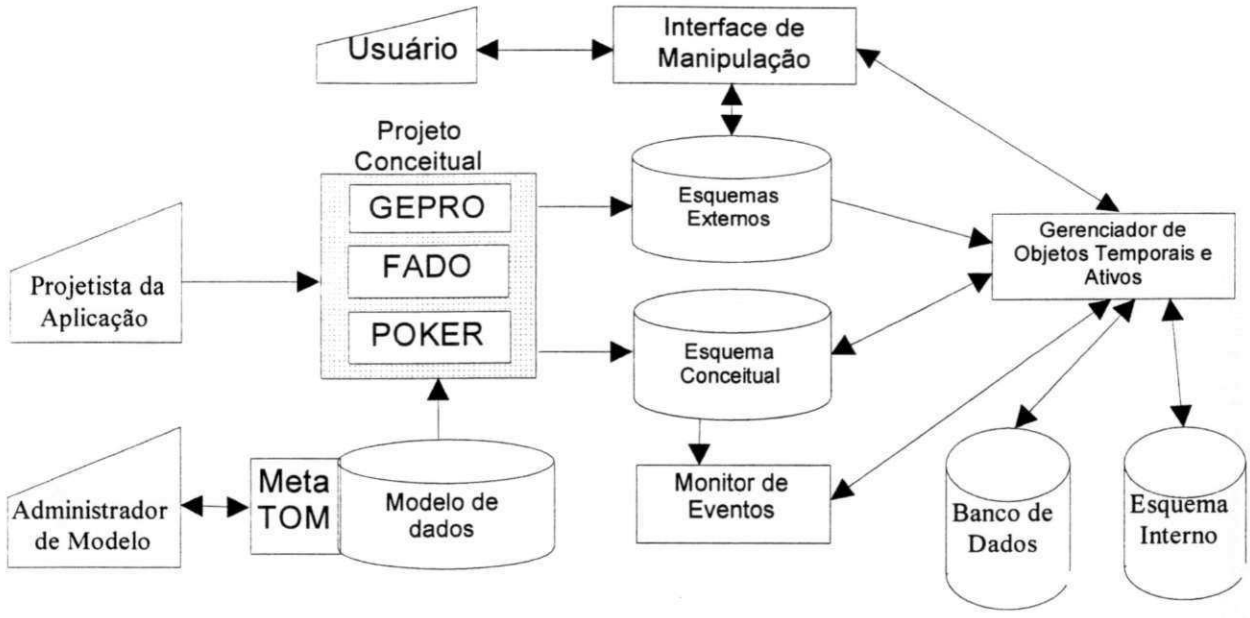


Figura 1-1 - Ambiente DYNAMO

A Figura 1.2, mostra o ambiente DYNAMO em forma de camadas. A primeira camada, a mais externa, apresenta as aplicações desenvolvidas pelo usuário. A segunda camada mostra o ambiente de desenvolvimento, neste nível encontram-se as metodologias (POKER [Schiel'89, Schiel'96] e FADO [Furtado'93]), as ferramentas de gerência e de desenvolvimento (GEPRO e CASE FADO) e o ambiente de consulta (ConTOM[Fernandes'94]), as aplicações a serem desenvolvidas usam os recursos disponíveis deste nível. O terceiro nível, apresenta o Gerenciador de Objetos Temporais e Ativos (GOTA), que faz o trabalho de conversão entre o ambiente de desenvolvimento e consulta e o gerenciador de banco de dados. É neste nível que fica o gerenciador de regras TOMRules[David'92] e o integrador de visões que gerenciará o acesso aos bancos de dados disponíveis ao ambiente. O quarto nível utiliza um gerenciador de banco de dados disponível no mercado.

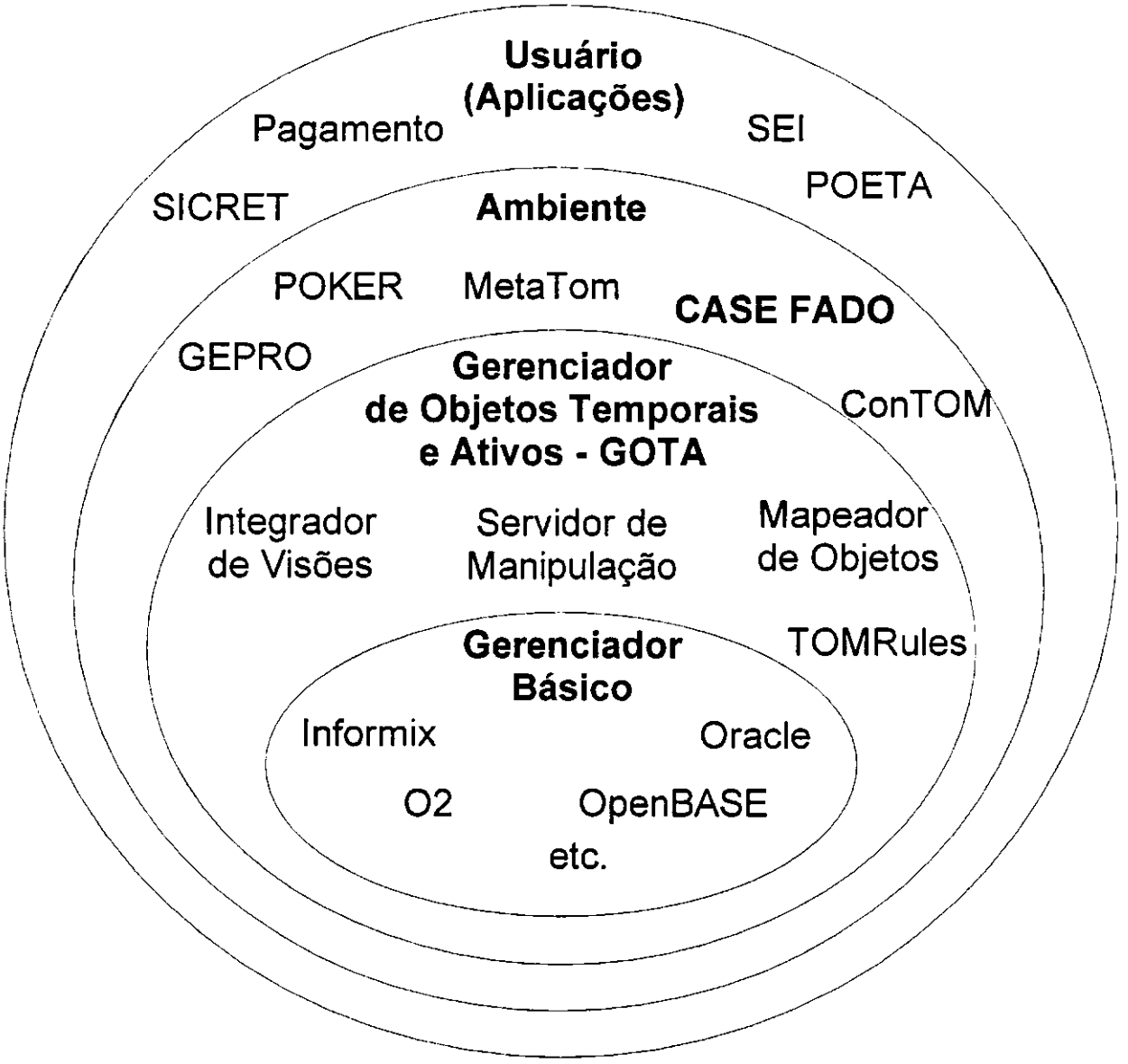


Figura 1-2 - Ambiente DYNAMO

1.3 Estrutura da Dissertação

Esta dissertação possui além desta introdução mais seis capítulos, os quais resumimos a seguir.

No Capítulo 2, mostraremos alguns aspectos de outras metodologias usadas e, como tal iremos tecer alguns comentários sobre os métodos mais conhecidos, como os de Booch, Coleman (Fusion), Jacobson (OOSE, Objectory), Rumbaugh (OMT), Shaler/Mellor e UML, apenas para situar o leitor no contexto dos principais métodos.

No Capítulo 3, apresentaremos um breve histórico das ferramentas CASE, as várias definições deste tipo de ferramenta, o que é um ambiente CASE, o que são ferramentas simples e multimétodos, o que é um repositório, afinal uma idéia da abrangência do estudo das

ferramentas CASE.

No Capítulo 4, descreveremos em detalhes a metodologia FADO, mostrando passo a passo como usar seus diagramas e formulários no desenvolvimento de um Sistema de Informação. Tentamos mostrar, de uma maneira resumida, as vantagens de se usar esta metodologia, cuja versão original, desenvolvida na UFPB como uma dissertação de mestrado[Furtado93], sofreu algumas modificações.

No Capítulo 5, mostraremos o desenvolvimento da CASE FADO, que utiliza-se da própria metodologia FADO no seu projeto, apresentando o contexto da ferramenta e todas as suas intenções, tentando visualizar todas as opções disponíveis. A plataforma de desenvolvimento da ferramenta foi o ambiente de desenvolvimento Delphi, e o ambiente de utilização PC/Windows.

Finalmente no Capítulo 6, teceremos as últimas considerações sobre o trabalho desenvolvido, indicando os resultados obtidos, as dificuldades encontradas e dando algumas sugestões sobre o desenvolvimento de trabalhos futuros.

2. Metodologias de Projeto OO

2.1 Introdução

No início dos anos 60, começava a surgir a necessidade de se construir projetos de software cada vez maiores, com uma complexidade crescente e em um tempo cada vez menor. A complexidade dos grandes projetos os tornava de difícil entendimento e seus requisitos eram freqüentemente difíceis de serem entendidos, resultando em especificações não confiáveis e ineficientes, documentação pobre e muita manutenção após sua liberação. Métodos e ferramentas associadas passaram a suportar o desenvolvimento de um projeto de software, mas forneciam pouco suporte para fazer frente aos desafios intelectuais de atender adequadamente aos grandes projetos. Isto ficou largamente conhecido como a "crise do software" [Penedo'88]. Da "crise do software" dos meados dos anos 60 para os dias de hoje, muitos conceitos, metodologias, linguagens, ferramentas e técnicas foram introduzidas com o objetivo de aperfeiçoar os processos de desenvolvimento de software e seus produtos.

A partir de maio de 1974, quando foi publicado no *IBM Systems Journal* o ensaio revolucionário de "Projeto Estruturado" por Larry Constantine, o mundo tomou conhecimento dessas novas idéias. Após esta publicação veio o lançamento de vários livros por Glenford, Yourdon e Jackson, que culminou com a idéia de Análise Estruturada, e também vieram várias publicações sobre este tema com Tom DeMarco, Chris Gane, Coad/Yourdon e muitos outros.

A análise estruturada ainda é muito utilizada, mas têm surgido idéias novas como, por exemplo, a Engenharia da informação[Martin'91], que também dá ênfase à modelagem dos dados, e o surgimento de um novo paradigma, a Orientação a Objetos, que começou sendo usado apenas na programação, sendo depois ampliado para análise e projeto. As primeiras publicações que propõem um desenvolvimento orientada a objetos (classe, objeto, encapsulamento, etc) vieram a público por volta de 1986 com [Booch'86], [Cox'86], [Jacky'86], [Meyer'88], [Shlaer/Mellor'88], e outros.

2.1.1 Histórico da Orientação a Objetos

Segundo [Page-Jones'97], diferente das muitas descobertas humanas, a Orientação a Objeto não surgiu de um momento para o outro. Em vez de uma inspiração do tipo "Eureka!" que algumas pessoas têm quando estão no chuveiro, ela é a combinação do trabalho de muitas pessoas em muitos chuveiros durante muitos anos. Os conceitos da orientação a objeto são como

vários afluentes que fluem quase que por acidente histórico para formar o rio da orientação a objeto.

Abaixo, também retirado do livro de Page-Jones, comentamos o trabalho de algumas pessoas que contribuíram, na teoria ou na prática, para a torrente orientada a objeto.

Larry Constantine pesquisou extensivamente os critérios fundamentais para um bom projeto de software [Constantine'68]. Ele foi uma das primeiras pessoas a sugerir que o software poderia ser projetado antes que fosse programado. Muitas das noções de Constantine têm demonstrado ser relevantes à orientação a objeto do mundo de hoje. Nesse livro é discutido a "conascença", um conceito fortemente baseado nos conceitos duráveis de acoplamento e coesão de Constantine.

Dahl e Nygaard introduziram várias idéias que agora são consideradas orientadas a objeto. O melhor exemplo é a idéia de classe, que apareceu pela primeira vez na linguagem Simula (Dahl e Nygaard, 1966) [Dahl/Nygaard'66].

Kay, Goldberg e seus colegas introduziram as primeiras versões da linguagem Smalltalk no Xerox Palo Alto Research Center, por volta de 1970 [Kay'69]. Essa pesquisa nos deu muitos dos conceitos que agora consideramos fundamentais à orientação a objeto (como mensagens e herança). Muitas pessoas ainda consideram a linguagem e o ambiente Smalltalk (agora mais conhecido como Smalltalk-80, [Goldberg/Robson'89] como a mais pura implementação da orientação a objeto da atualidade.

Dijkstra tem-nos causado perpétua culpa por mais de trinta anos. Em seu primeiro trabalho, ele propôs as idéias da construção do software em camadas de abstração com rigorosa separação semântica entre camadas sucessivas. Isso representa uma poderosa forma de encapsulamento, que é fundamental para a orientação a objeto.

Liskov fez progressos significativos na década de 1970 na teoria e implementação de tipo abstrato de dados (ADT), que constitui o fundamento para a orientação a objeto. Um dos resultados mais destacados do trabalho de Liskov foi a linguagem CLU, que apoia solidamente a noção de ocultação de representações de dados internos.

Em um trabalho memorável, Parnas escreveu sobre os princípios da boa construção de software modular [Parnas'72]. Embora os construtores da orientação a objeto transcendam aos módulos procedimentais tradicionais, muitos dos princípios básicos de Parmas sobre ocultamento de informações ainda são aplicáveis em sistemas orientados a objeto.

Ichbiah e seu grupo desenvolveram a linguagem de Programação "Green", que o Departamento de Defesa dos Estados Unidos adotou como a linguagem ADA (agora conhecida

como Ada-83)[1]. Duas das características da Ada-83 (genericidade e pacote (*package*)) estão bem no coração da orientação a objeto. Uma versão recente da linguagem, a Ada-95, apoia totalmente a orientação a objeto.

A linguagem C++ tem uma genealogia interessante. Havia antigamente a linguagem BCPL, de Martin Richards [Richards/Whitby-Strevens'80]. Isso gerou a linguagem B, uma abreviação (!) de BCPL. B gerou C, e C (através do trabalho de Stroustrup) gerou a linguagem orientada a objeto C++. Conforme descrito em [Stroustrup'91, p. 4], a "geração de C++" parece ser um acontecimento eventual:

> C++ foi projetada inicialmente para que o autor e seus amigos não tivessem de programar em linguagem de montagem assembler, C ou várias linguagens modernas de alto nível. Sua finalidade principal é tornar a escrita de bons programas mais fáceis e agradáveis para o programador individual. Nunca houve um projeto de C++ no papel; projeto, documentação e implementação ocorreram simultaneamente.

Como a orientação a objeto da linguagem C++ foi moldada sobre linguagens anteriores, não orientadas a objeto, e de baixo nível, sua sintaxe nem sempre é pura. Contudo, C++ é atualmente a linguagem orientada a objeto mais amplamente usada. Sua antecessora, a linguagem C, deu-lhe a portabilidade para muitas máquinas e sistemas operacionais e assim aumentou muito a popularidade da programação orientada a objeto. Nesse aspecto, a contribuição de Stroustrup foi imensa.

A força do trabalho de Meyer é a fusão das melhores idéias da Ciência da Computação com as melhores idéias da orientação a objeto. O resultado foi uma linguagem e ambiente chamado Eiffel [Meyer'88]. Eiffel é uma raridade no mundo do software, pois atrai simultaneamente acadêmicos, engenheiros de software e aquelas outras pessoas nas "trincheiras" que precisam de um código robusto. Qualquer que seja a linguagem orientada a objeto que você escolha, deverá aprender os conceitos nos quais se baseia a Eiffel se quiser tornar-se um verdadeiro profissional orientado a objeto. No Capítulo 10, do livro de Page-Jones, são tratados os termos invariantes de classe, précondições e pós-condições, o fundamento do "projeto por contrato" de Meyer.

Hoje, como a orientação a objetos está mais madura e existe um grande interesse pelas técnicas de análise e projeto orientadas a objeto, apresentaremos a seguir uma pequena amostragem das metodologias existentes até a confecção deste trabalho, sem nenhuma pretensão de mostrar conhecimento absoluto desta vasta área. Afinal este trabalho deve dar uma visão

1 Ada é uma marca registrada do governo dos Estados Unidos.

geral dos métodos para que a apresentação da ferramenta CASE proposta possa ser compreendida.

A discussão a seguir tem como base o trabalho de Martin Fowler publicado como tutorial no [OOPSLA'95].

2.2 Visão Geral dos Métodos

Os métodos de análise e projeto de sistemas são uma maneira sistemática de desenvolver uma aplicação informatizada. No desenvolvimento de sistemas é comum usar várias técnicas para que se possa cobrir todo o projeto, basicamente porque cada técnica dá maior ênfase a um aspecto do projeto e negligencia outro. Por exemplo, o modelo Entidade-Relacionamento (ER) descreve os dados de um sistema e não faz qualquer referencia aos processos. Normalmente um método irá usar várias técnicas semelhantes para cobrir todos os aspectos de um projeto. Técnicas não são exclusivas de um método, a mesma técnica pode ser usada por vários outros métodos. A técnica básica é a mesma, mas cada metodologia introduz alguns novos conceitos de modelagem e novas notações. [Shlaer/Mellor'88] e [Rumbaugh'91] são dois métodos usando diferentes dialetos da técnica ER. Eles diferem em alguns conceitos (não existe o conceito de agregação em Shlaer/Mellor), e na notação (a notação de cardinalidade é diferente).

As técnicas de modelagem mostram a sua maneira de ver o sistema, enfatizando alguns aspectos do sistema e negligenciando outros. Comumente podemos considerar três maneiras de ver um sistema: estrutura, comportamento, e arquitetura (Figura 2-1). A Visão Estrutural (visão dos Dados) descreve a parte estática do sistema, negligenciando os processos. Nos sistemas que usam banco de dados isto corresponde ao esquema do banco de dados; em OO isto corresponde as classes, atributos e os relacionamentos entre elas. Diagramas Entidade Relacionamento e o Diagrama Estático da Metodologia FADO, são um exemplo de visão estrutural. A Visão Comportamental descreve como o sistema muda, é a descrição de como irá rodar. O diagrama de transição de estados é um exemplo de visão comportamental. Usando a visão estrutural e a visão comportamental pode ser feita uma descrição completa do sistema, mas esta descrição pode ser complexa e de difícil entendimento. Assim a Visão Arquitetural pode ser usada para quebrar o sistema em subsistemas, para que se possa a partir desses subsistemas usar a Visão Estrutural e a Visão Comportamental para desenvolve-los.

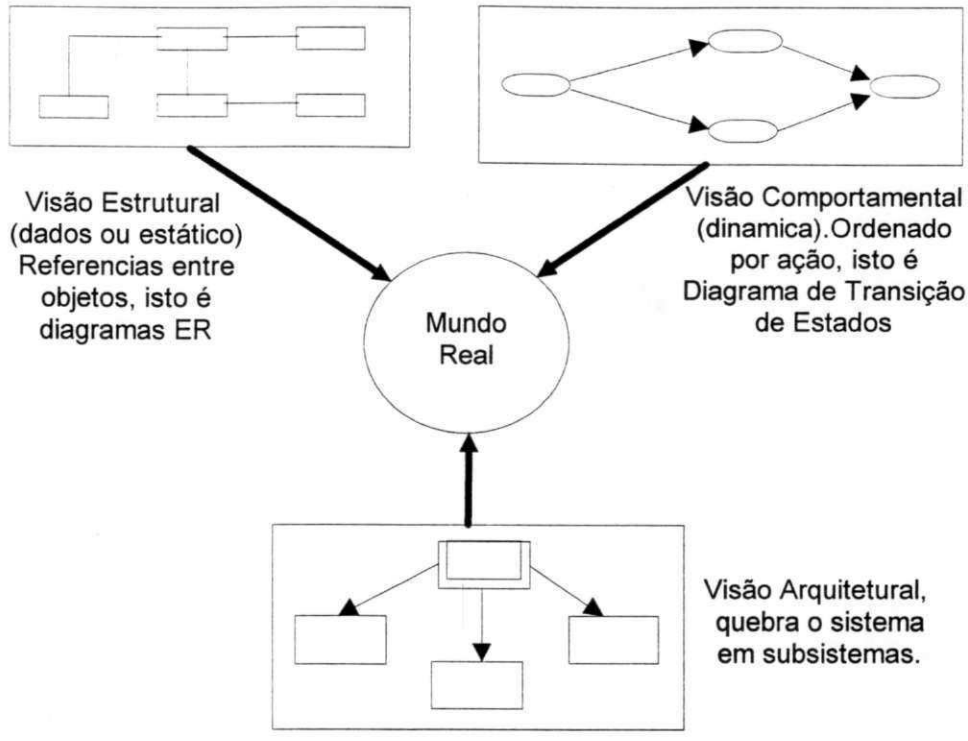


Figura 2-1 - Três visões de um sistema

Existem dois aspectos-chave em uma técnica: seu rigor e a sua compreensibilidade. Uma técnica rigorosa é aquela que não permite qualquer ambigüidade em sua interpretação. Isto é frisado ao máximo em métodos formais. Compreensibilidade é freqüentemente negligenciada mas é igualmente importante. A razão para isto é que uma técnica que é difícil de entender é difícil de ensinar e deste modo torna mais difícil novas pessoas entenderem e utilizarem corretamente o modelo. Isto é importante, visto que ela reduz as chances dos usuários fazerem contribuições efetivas para o processo de análise e avaliação do modelo. Muitos modeladores de dados encontram nos diagramas ER uma visão muito compreensível da estrutura e diagramas ER podem ser facilmente definidos de uma maneira totalmente formal.

Outro aspecto de uma técnica é a sua expressividade. Algumas técnicas são capazes de dizer mais coisas e assim poderem expressar mais conceitos que uma técnica menos expressiva. Expressividade contudo não é totalmente uma virtude. Técnicas mais expressivas têm mais conceitos e são assim mais difíceis de aprender. Assim uma técnica menos expressiva perde em alguns aspectos, mas a falta de conceitos pode ser o valor pago para uma aprendizagem rápida. Uma boa maneira é balancear expressividade e simplicidade.

Assim um método consiste de um número de diferentes técnicas combinadas para formar um modelo coerente de um sistema. Assim como existem técnicas de modelagem, um conjunto de heurísticas de modelagem são necessárias para descrever como construir esse modelo. O

termo heurística foi usado por que sempre existe alguma coisa solta a ser descrita, mas de muita utilidade.

2.2.1 Os Métodos

A escolha de um método não é fácil, e conhece-lo mais difícil. Existem muitos livros textos sobre métodos e idéias novas e importantes são publicadas constantemente.

Uma pesquisa publicada pela revista ADT - Aplication Development Trends (maio/97) [Developers'97], realizada pela SPG - Software Productivity Group, mostra o perfil de uso dos diferentes métodos em empresas americanas (Figura 2-2). A metodologia caseira, segundo essa pesquisa, é a mais usada, mostrando que a grande maioria dos usuários monta a sua própria metodologia a partir de outras. Isto quer dizer também que o usuário não está satisfeito com as metodologias existentes.

Segundo a mesma pesquisa, a UML (Unifield Modeling Language) promete muitas adesões. Esta nova metodologia patrocinada pela Rational Rose, incorpora os métodos de Booch, Rumbaugh e Jacobson..

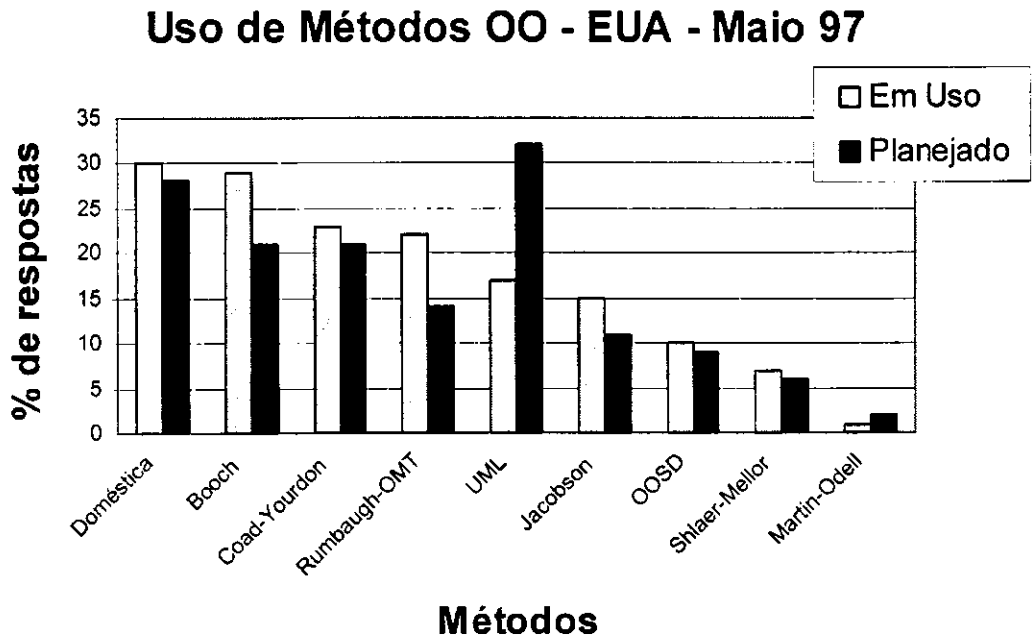


Figura 2-2 Tabela de uso dos métodos

2.3 Visão Estrutural dos Métodos

A visão estrutural de um sistema concentra-se na descrição dos tipos de objetos do sistema e dos vários tipos de relacionamentos estáticos existentes entre eles. Os três principais tipos de relacionamentos existentes são: associações ou relacionamentos (um cliente pode alugar um certo número de filmes), subclasse (generalização/especialização) (uma enfermeira é um tipo de pessoa), e agregação (um motor é parte de um automóvel). Todos os vários métodos OO usam diferentes (e muitas vezes conflitantes) terminologias para estes conceitos. Isto é extremamente frustrante mas inevitável. Para se ter uma idéia dos termos utilizados foi montada a Figura 2-3.

2.3.1 Classes

A noção de classe é comum e elas são representadas como uma caixa em um diagrama, embora o desenho da caixa varie de método para método.

Termos usados na visão estrutural

FADO	Booch	Coad	Jacobson	Odell	Shlaer/ Mellor	Rumbaugh
Classe	Classe	Classe e Objeto	Objeto	Classe de Objeto	Objeto	Classe
Relacionamento	Associação	Conecção de Instancia	Associação de Conhecimento	Associação	Relacio-namento	Associação
Generalização	Herança	Gener/ Especia.	Herança	Subclasse	Subtipo	Generaliza-ção
Agregação	Agregação	Parte/Todo	Consiste de	Composição	Não tem	Agregação
Agrupamento	--	--	--	--	--	Agregação

Figura 2-3 Termos usados nas metodologias

2.3.2 Associação ou Relacionamento e Atributos

Associação representa o relacionamento entre instancias de classes de objetos (uma pessoa trabalha para uma companhia, uma companhia tem escritórios, etc). Uma interpretação exata de uma associação depende do método. Na análise elas representam a responsabilidade pelo conhecimento. Assim uma associação entre pedido e cliente é interpretado pela ligação de pedido com cliente, (Figura 2-4).

Convencionalmente é assumido que associações multivaloradas são conjuntos (ex: os filhos de uma pessoa constituem um conjunto de pessoas). Em outros métodos, como em FADO, esses conjuntos são chamados de grupos e a associação é um agrupamento (Seção 2.3.5).

Algumas linguagens Orientadas a Objeto (OO) e a Metodologia FADO permitem associações entre instâncias de objetos e classes. São as variáveis de classe do Smalltalk e os relacionamentos de classe do TOM.

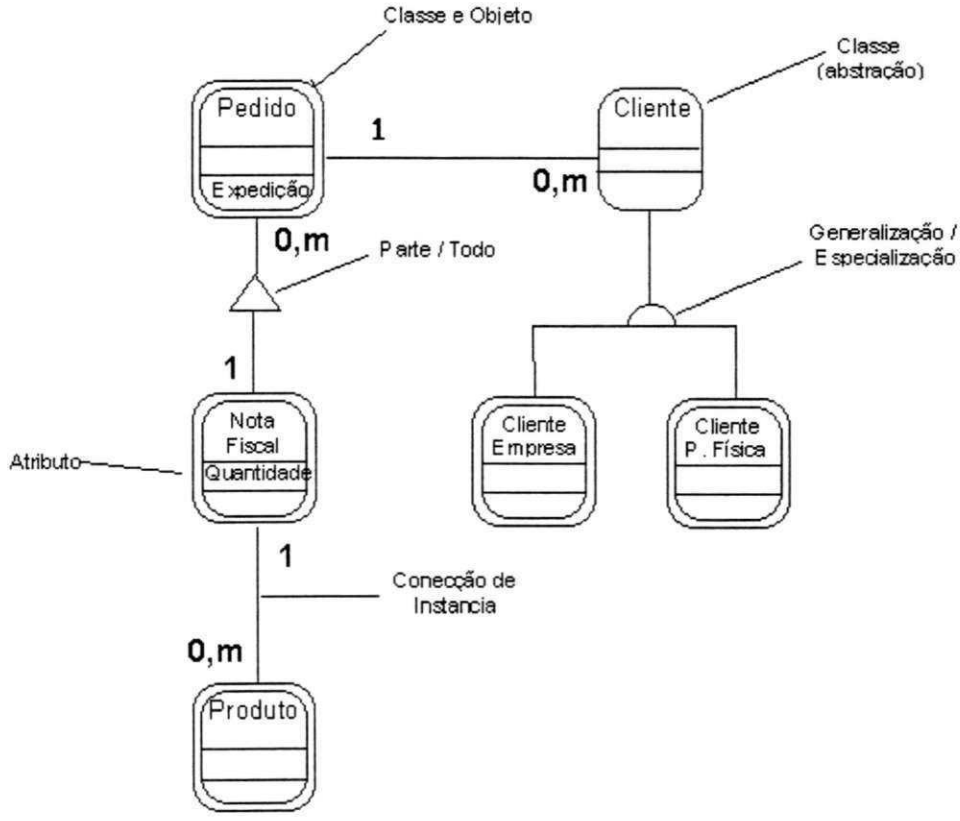


Figura 2-4 - Modelo do Coad de OOA usando classe e objeto, nível de estrutura e atributo.

Às associações é atribuída uma cardinalidade, que corresponde à noção de obrigatoriedade, opcional, 1-muitos, muitos-muitos. Cada método usa uma notação particular para indicar a cardinalidade. Nos métodos estruturados o estilo *pé-de-galinha* (usado por Odell) foi o começo para uma padronização, mas a comunidade OO jogou tudo fora. Agora com a associação do três grandes estudiosos de metodologias (Booch, Rumbaugh e Jacobson) a idéia de padronização despertou com muita força. A guerra agora vai ser quem será aceito como padrão. A OMG já registrou três propostas de padronização dos métodos, uma da Rational Rose, chamada OTUG (Object Technology Users Group) [Rational], que se ampara na UML (Unifield Modeling Language) coordenada pelos seus três estudiosos; outra pela Platinum, denominada OML (Open Modeling Language) comandada por Mary Loomis e James Odell, e outra pela

IBM, chamada COMN (Common Object Modeling Notation) pilotada por Meilir Page-Jones [IBM]. Vale salientar que em maio de 97 a Platinum anunciou sua saída do projeto OML e aderiu à bandeira da UML da Rational.

A cardinalidade representa os valores de um relacionamento. Os valores mais comuns para cardinalidade são: mínimo zero e um; para os valores máximos, um ou muitos. Estes valores permitem quatro combinações apresentadas na Figura 2-5.

Leia da Esquerda para Direita	Um A é sempre associado a um B	Um A é sempre associado com um ou muitos de B	Um A é associado com zero ou um de B	Um A é associado com zero ,um, ou muitos de B
Booch (primeira edição)	A 1 B	A + B	A ? B	A * B
Booch (2a ed., deve ser unidirecional)	A ● 1 B	A ● 1...N B	A ● 0...1 B	A ● N B
Coad	A 1 B	A 1,m B	A 0,1 B	A 0,m B
Jacobson (unidirecional)	A (1) ▶ B	A (1...M) ▶ B	A (0...1) ▶ B	A (0...M) ▶ B
Odell	A ─|| B	A ─< B	A ─○| B	A ─○< B
Shlaer / Mellor	A ─> B	A ─>> B	A C ─> B	A C ─>> B
Rumbaugh	A ─ B	A 1+ B	A ─○ B	A ─● B
FADO	A (1,1) B	A (1,*) B	A (0,1) B	A (0,*) B
UML	A 1 B	A 1..* B	A 0..1 B	A 0..* B

Figura 2-5 - Diferença de notação de cardinalidade em diferentes métodos.

2.3.3 Agregação

Agregação é o processo de construir novos objetos a partir da composição de objetos existentes. Relacionamentos de agregação ("parte de" ou "tem um") são usados por muitos métodos. Ela indica um relacionamento parte/todo (ex: um martelo é feito de cabeça e cabo).

Formalmente, uma agregação é definida como um subconjunto do produto cartesiano dos objetos componentes e sua identificação é determinada pelas identificações de seus componentes. (Figura 2-6).

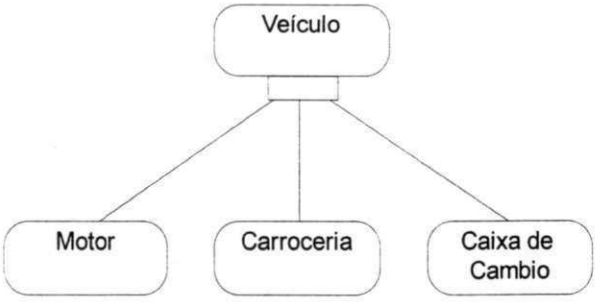


Figura 2-6 - Exemplo de agregação

2.3.4 Generalização/Especialização

Generalização/Especialização é um conceito dos modelos semânticos de dados usados na modelagem OO.

Especialização é o processo de selecionar uma subclasse a partir de uma classe de objetos para acrescentar algumas propriedades específicas das instâncias da subclasse. Por exemplo podemos definir a subclasse Horista a partir da classe Trabalhador. Isto corresponderia ao conceito de herança na programação OO. Generalização é o processo inverso, suprimindo as diferenças entre as classes, identificando suas propriedades comuns para gerar uma superclasse é usado o predicado "é-um". Um exemplo de Generalização/ Especialização está na Figura 2-7.

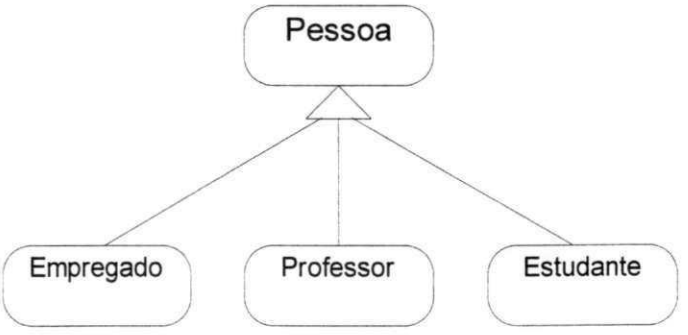


Figura 2-7 - Exemplo de generalização/especialização

2.3.5 Agrupamento

Agrupamento é uma abstração usada para representar um conjunto de objetos do mesmo tipo. É representada pelo predicado "é-elemento". Por exemplo, grupos de objetos da classe Árvore formam instâncias da classe Floresta. Formalmente, a classe de grupo é um subconjunto do conjunto das partes da classe dos elementos. A metodologia FADO faz uso deste tipo de abstração. A Figura 2-8 apresenta um exemplo de agrupamento.

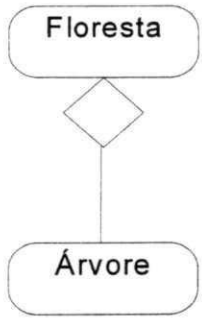


Figura 2-8 - Exemplo de agrupamento

2.3.6 Regras

Uma ampla visão de regras permite outra percepção da estrutura. Dois trabalhos importantes nesta área foram escritos por [Odell'93] e [Graham'93]. Odell discute regras estruturais e comportamentais e como elas são definidas. Graham adicionou regras ao modelo estrutural e discute muitos dos princípios de inteligência artificial (IA), adicionando regras *fuzzy* à discussão.

Na metodologia FADO, regra é a associação entre eventos e *triggers*, determinando quais eventos irão ativar quais *triggers* e em que condições. Na Figura 2-9 mostramos a estrutura e comportamento do modelo de regras FADO.

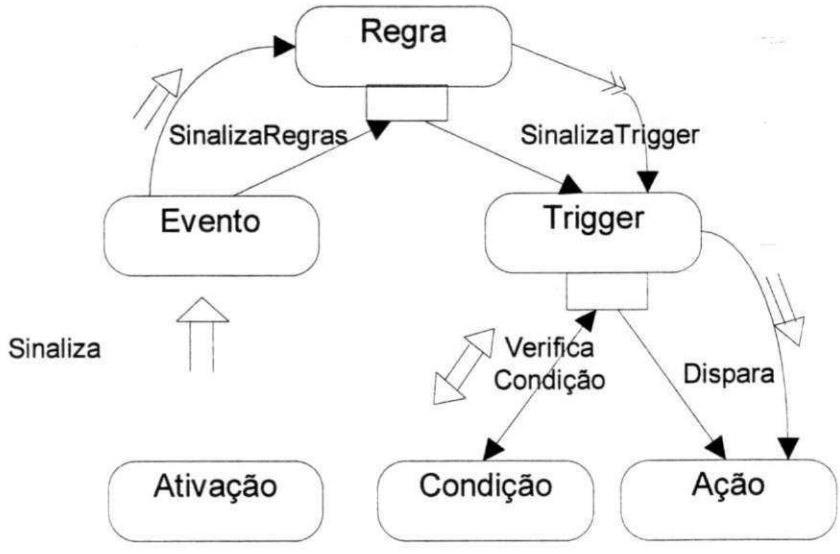


Figura 2-9 - Modelo de regras em TOM

2.4 Visão Comportamental dos Métodos

Uma técnica que tem recebido uma atenção especial na modelagem comportamental de uma aplicação é a dos diagramas de transição de estados. Esse tipo de diagrama é formado por caixas que representam os estados do sistema sendo modelado e conectadas por linhas que representam transições entre esses estados. Cada transição deve ser ligada a um evento. Quando

um evento ocorre, o sistema reage dependendo de qual estado ele esteja. Então é marcada uma transição para um novo estado. Isto traz muitas vantagens: esse tipo de diagrama tem uma base formal bem trabalhada e é "executável". Embora esta arquitetura geral aplique-se para todos os diagramas de transição de estados, nos métodos que os usam eles variam em detalhes. Não importa o desenho de um diagrama de estados, eles tem a desvantagem de solicitarem todos os estados do sistema a ser definido como nós em um diagrama. Isto é excelente para pequenos sistemas, mas para grandes sistemas existe um crescimento exponencial no número de estados. Esta "explosão de estados" deixa o diagrama de transição de estados muito complexo dificultando o uso prático.

Para combater o problema da "explosão de estados", os métodos orientados a objetos definiram um diagrama de transição de estados para cada classe. Isto elimina o problema da explosão de estados, deixando cada classe com um diagrama compreensível de transição de estados. Contudo, cria um outro problema, que é a visualização global do comportamento de um sistema a partir de diagramas individuais de classes. Este problema levou ao desenvolvimento de técnicas de interações e de modelagem baseada em eventos, que leva o modelador a descrever como várias classes cooperam em um sistema real.

2.4.1 Variedade de Transições de Estados

Uma das principais diferenças entre os vários tipos de diagrama de transição de estados é onde as ações são habilitadas. Na abordagem do modelo de [Mealy'55], as ações estão ligadas à transição. Quando uma transição ocorre uma ação deve ser executada. Então o sistema fica aguardando em seus vários estados. O modelo de [Moore'56], liga ações a estados. Quando uma mudança de estado é iniciada uma ação é executada. Um depende da transição, o outro depende do estado. Existe uma diferença sutil, mas isto é muito importante para a produção de diferentes diagramas de uma mesma situação.

Esta diferença é ilustrada pela comparação dos diagramas de transição de estados das Figuras 2.10 e 2.11. Os dois diagramas considerados inicialmente são o de [Booch'91] e uma versão de Shlaer/Mellor. [Booch'91] usa o modelo de [Mealy'55] enquanto Shlaer/Mellor usa o modelo de [Moore'56].

Outro nível de sofisticação para diagrama de transição de estados foi descrito em [Harel'87] e aceito por Rumbaugh, e depois por [Booch'94], (Figura 2.12). O mais perceptível nesta extensão é a noção de super-estados e de sub-estados, que pode ser visto na versão do diagrama de Rumbaugh. Este admite um comportamento comum através dos estados capturados (ex: eventos Suspender e Abandonar). Neste estilo existe uma diferença entre *ações* que são

consideradas instantaneamente pelo contexto do modelo, a as *atividades* que levam um tempo significativo. *Ações* são ligadas a transições e *atividades* a estados. Além disto, *ações* podem ser ligadas a entrada de um estado e a saída de um estado.

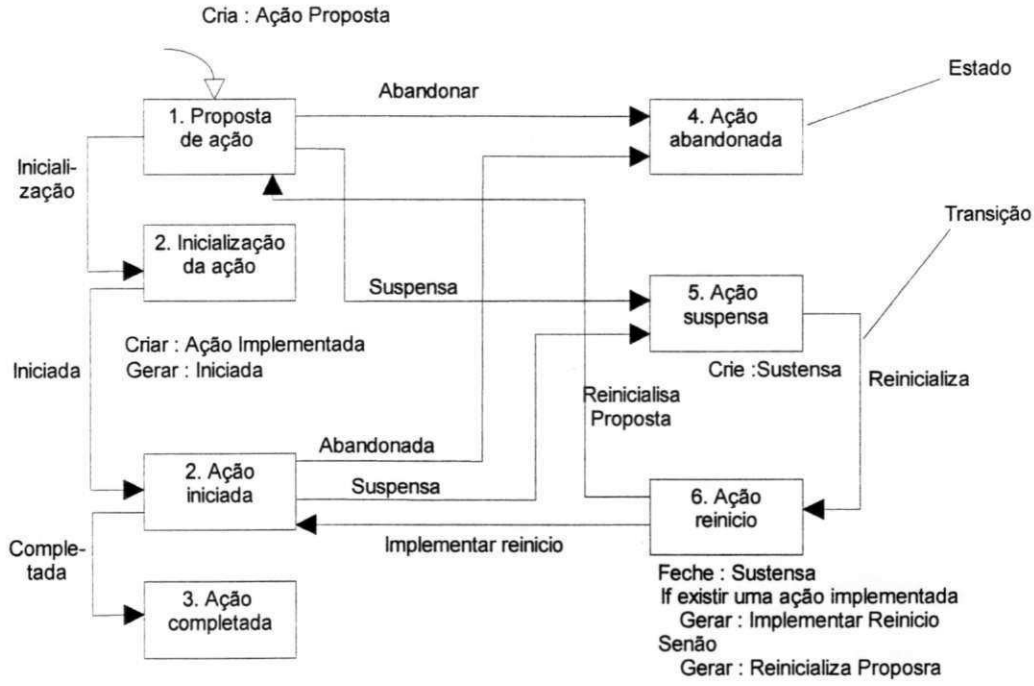


Figura 2-10 - Diagrama de transição de estados de Shlaer/Mellor

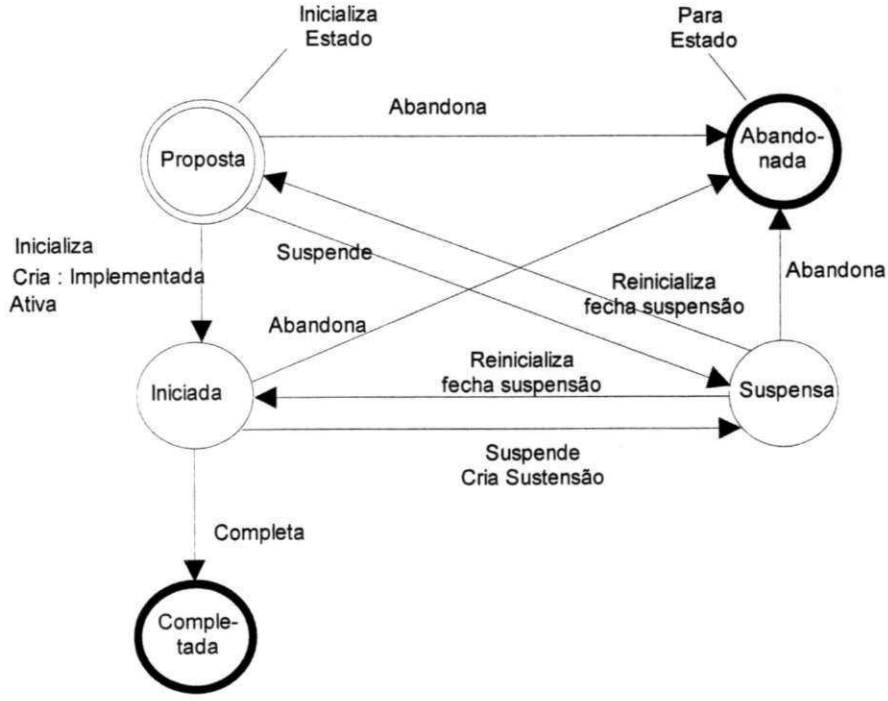


Figura 2-11 - Diagrama de transição de estados de Booch' 91

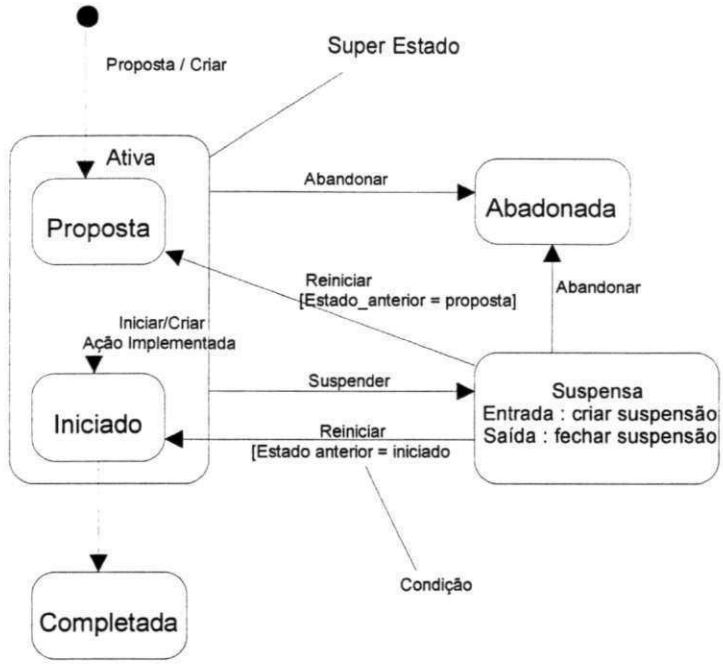


Figura 2-12 - Diagrama de transição de estados de David Harel (também usado por Rumbaugh e Booch).

2.4.2 Comportamento entre Classes

A limitação do diagrama de transição de estados a uma classe deve-se à eliminação do problema da explosão de estados, mas isto também tem seus problemas. Embora o comportamento de uma classe fique fácil de visualizar, é difícil visualizar o comportamento do sistema como um todo a partir do comportamento individual das classes. Existem três maneiras básicas de abordar isto. A primeira é não forçar o problema como um todo numa modelagem comportamental. Esta abordagem descreve apenas o comportamento interclasse de uma maneira arquitetural. Nesta abordagem cada mensagem e evento enviado desta classe é descrito, mas não se tenta descrever a seqüência de como essas mensagens são enviadas. É importante ressaltar que isto não faz nenhum efeito no rigor da abordagem, o diagrama de transição de estados para cada classe descreve completamente o comportamento de um sistema complexo.

A segunda maneira é a abordagem baseada em interações, que tenta visualizar o comportamento interclasse. Abordagem baseada em interações representa um número de objetos e mensagens que são passadas entre eles em uma seqüência. Isto pode ser feito apresentando os objetos como ícones, as mensagens como setas e uma numeração para apresentar a seqüência (como o diagrama de objetos de Booch ou nível de serviço de Coad ou o Diagrama contextual da FADO). A outra abordagem é apresentar os objetos como linhas verticais e as mensagens como linhas horizontais, mostrado em seqüência de cima para baixo (como no diagrama de interações de Jacobson e Booch (Figura 2-13), e no diagrama de eventos de Rumbaugh (Figura

2-14)). Estas notações dão uma boa visão da interação entre os objetos. Contudo, em ambos os casos a seqüência de ações pode ser descrita mas não pode ser definida. Assim, quando usados desta maneira, eles agem como suporte ao diagrama de estados. [Jacobson'92], em particular, vai mais longe quando usa pseudocódigo em conjunto com o diagrama de interações para definir um comportamento detalhado. Quando isto é feito, a necessidade do suporte para o diagrama de estados é reduzido, podendo ser eliminado.

O conceito de interações permite descrever o comportamento inter-objeto, mas tem desvantagens. O primeiro problema é a falta de uma construção diagramática que seja rica o bastante para ser computacionalmente completa. O segundo problema está no fato de que interações são essencialmente algoritmos e descrevem uma simples "thread" de controle, não existe maneira de combinar interações para mostrar uma visão geral do comportamento de um sistema, a não ser via uma rede de Petri.

Este tipo de problema ocorre na abordagem baseada em eventos, como no diagrama de eventos de Odell. O diagrama de eventos, criado por John Edwards, começa por uma diferença básica do diagrama de eventos; em vez de tratar os estados como uma construção de blocos, ele foca-se nos eventos. Eventos são produzidos por operações que são conectadas por regras de disparos (*trigger*) : uma estrutura não inteiramente diferente para o Diagrama de Fluxo de Dados e Ações de Shlaer/Mellor ou o Diagrama Dinâmico da metodologia FADO. O diagrama de eventos é mais usado para (potencialmente) descrever inteiramente o comportamento de um sistema. Embora um diagrama de eventos possa ser usado para descrever uma transição de estado estilo ação ou uma interação da mesma maneira que um pseudocódigo, é poderoso o bastante para ser capaz de descrever o comportamento com muitas diferentes interações de "*threads*" de controle.

Na metodologia FADO, o Diagrama Contextual apresenta o comportamento do sistema mostrando o envio de mensagens entre as classes através de setas. Contudo não apresenta uma seqüência cronológica no envio dessas mensagens, deixando isto para o Diagrama Dinâmico, que além da seqüência cronológica apresenta o comportamento detalhado da aplicação.

Portanto aqui estão quatro maneiras de ver o comportamento da modelagem: baseada em procedimentos, baseada em estados, baseada em interações e baseada em eventos. Como sempre na discussão de modelagem, o mais importante, e que não deve ser esquecido, é que não existe uma técnica de modelagem que seja melhor para todos os problemas. O diagrama de eventos é superior quando o comportamento de grandes sistemas precisa ser definido através de muitas classes.

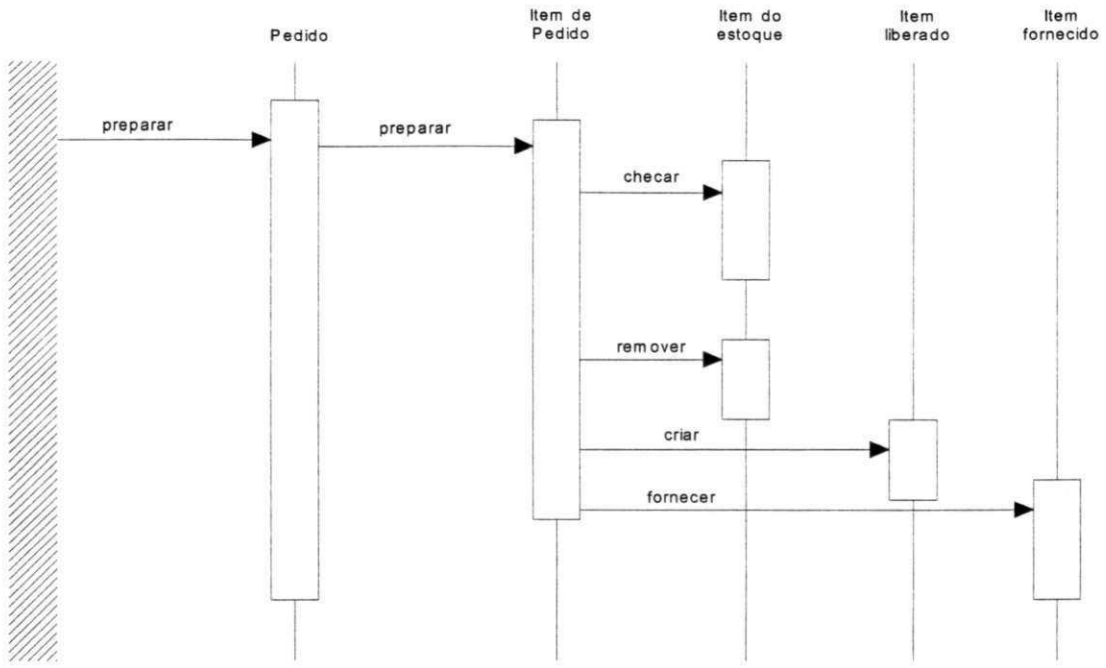


Figura 2-13 - Diagrama de interação de Jacobson.

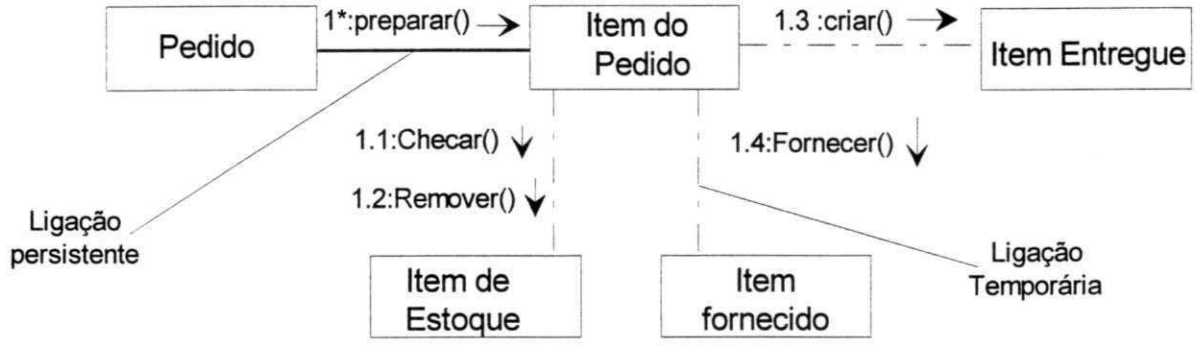


Figura 2-14 - Diagrama de fluxo de eventos de Rumbaugh

2.5 Visão Arquitetural dos métodos

A visão arquitetural consiste em quebrar grandes sistemas em subsistemas. Isto tem sua funcionalidade por que funções de alto nível são quebradas em subfunções, estas subfunções são quebradas novamente em novas subfunções, e assim se repete até uma função simples, (idealmente até 7 ± 2 quebras). Por isso, torna-se difícil não seguir nenhuma orientação em um projeto. Por exemplo um fluxo de dados de um projeto serve para descrever não somente os módulos, mas também os dados que são passados entre eles. Isto permite que o modelador

concentre-se no fluxo de dados de um modelo particular para reduzir o acoplamento. Com a chegada da orientação a objetos, foi introduzida uma nova e diferente maneira, a decomposição em objetos, deixando de lado a decomposição em funções. O termo visão arquitetural cobre as duas abordagens, com métodos para a obtenção de decomposição funcional ou decomposição baseada em objetos.

2.5.1 Decomposição Funcional

A análise tradicional faz grande uso do diagrama de fluxo de dados. Um sistema é descrito pelas funções e armazenamento de dados com fluxo de dados conectando-os. Subfunções são quebradas em outras até que um nível básico seja encontrado. Nos últimos tempos ocorreram muitos debates para saber se este tipo de decomposição funcional podia ser usado para objetos. Agora os debates parecem ter sido vencidos pelos que acham que não pode. Decomposição funcional faz a separação entre dados e processos, que é geralmente vista como incompatível com objetos.

Somente um método OO discute diagrama de fluxo de dados o de [Rumbaugh'94]. Esta parte da abordagem de Rumbaugh recebeu uma severa crítica dos que seguem a OO, e dizem que o diagrama de fluxo de dados deve ser ignorado.

2.5.2 Decomposição em Objetos

A abordagem de decomposição em objetos sugere a quebra do sistema por agrupamento de classes, isto é, as classes que possuam as mesmas características devem ficar no mesmo grupo. Se nós imaginarmos um grande modelo estrutural nós poderemos imaginá-lo em grandes blocos, onde as classes de cada bloco estão mais interconectadas com as classes deste bloco do que com as de outro bloco. Esses pedaços são chamados de domínios, subsistemas, ou categorias de classe.

Voltando a atenção para esses domínios, nós poderemos observar como eles interagem em razão da estrutura do sistema em alto nível: isto é muito importante para grandes sistemas. Uma coisa que nós podemos fazer é observar todas as mensagens passada entre domínios, assim como no modelo do subsistema de comunicação de Shlaer/Mellor.

A abordagem usada por Booch é centrada na visibilidade. Primeiro, para uma classe fazer uso de outra classe (enviar mensagem para, ter uma lista de parâmetros numa operação), esta classe deve ser visível. Booch controla visibilidade pelo nível do domínio. Se um domínio tem visibilidade em outro domínio, então as classes do primeiro domínio podem usar classes do

segundo domínio. O propósito do projeto é limitar a visibilidade tanto quanto possível, reduzindo as dependências no sistema.

Dentro de um domínio, os classes podem ser públicas ou privadas. Classes públicas são vistas por aqueles domínios que possuem a sua visibilidade, classes privadas somente podem ser usadas por classes dentro do mesmo domínio. Domínios podem ser globais, no caso em que todos os outros domínios devam ter a visibilidade dele. Isto é necessário para componentes gerais assim como inteiros, strings, e coleções.

Usando este esquema, a interface de um domínio é a união de interfaces de todas as classes públicas em um domínio. É possível que esta interface fique muito generalisada. Pode ser que diferentes clientes deste domínio precisem de diferentes interfaces. Nestes casos a noção de contrato de [Wirfs-Brock'90] é bastante útil. Um contrato é essencialmente uma interface publica para um domínio. O contrato lista as características que estão disponíveis. Cada característica é implementada em um domínio. Um domínio pode ter mais de um contrato. [Booch'91] é o mais preocupado com isto, mas as descrições das suas técnicas são tremendamente sucintas.

2.5.3 Arquitetura Externa

A combinação de classes e subclasses deve fazer sentido para a obtenção de um sistema efetivamente funcionando, mas isto não torna visível o uso do sistema, isto é, não se consegue "ver" o sistema. [Jacobson'92] causou grande impacto com a introdução de cenários (ele o chama de "use case"). De fato, quase todos os modeladores de métodos estruturados ou de métodos orientados a objeto, têm usado cenários de uma maneira informal. O feito de [Jacobson'92] foi elevá-lo a uma posição formal no processo. Muitos dos usuários de cenários a vêem como uma técnica para descobrir objetos.

Cenários também fornecem uma maneira de descrever a visão externa de um sistema e suas interações com o mundo. Assemelham-se desta maneira a um diagrama de contexto em uma abordagem tradicional. Nesta representação, o mundo exterior é representado por "atores". "Atores" podem ser responsáveis por várias pessoas, ou outros sistemas de computador. A ênfase na função é importante: uma pessoa pode acionar muitas funções, e uma função pode ativar muitas pessoas. Cenários são tipicamente interações que os "atores" fazem com o sistema. A Figura 2-15 mostra um exemplo de cenário.

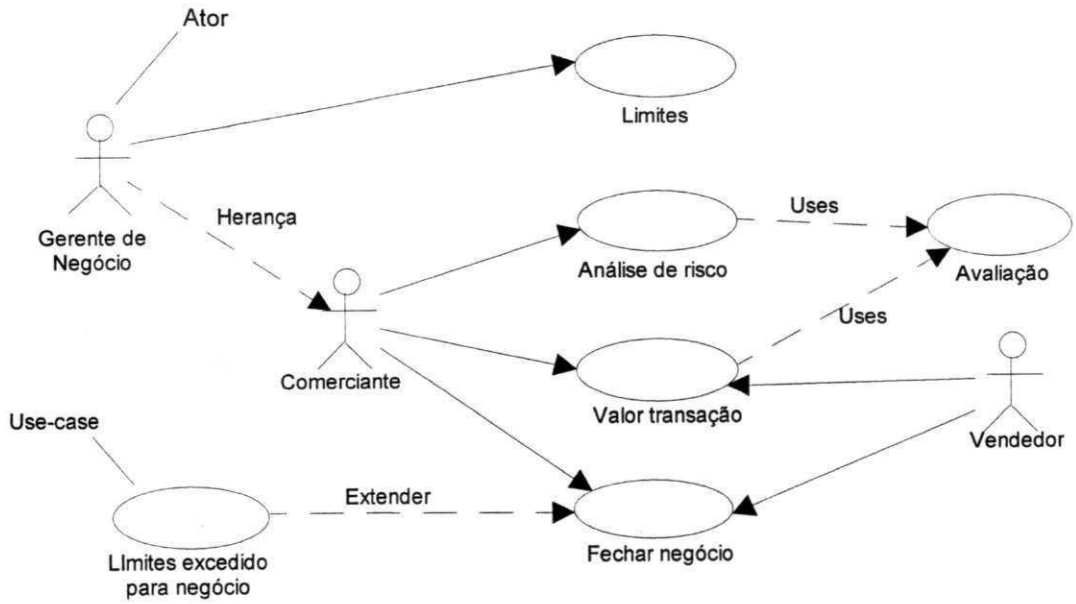


Figura 2-15 Exemplo de um cenário

3. Tecnologia CASE

3.1 Introdução

O ramo da Engenharia de Software chamada tecnologia CASE é a responsável pela automação das atividades ocorridas no processo de desenvolvimento de software. O objetivo do aparecimento destas ferramentas está relacionada com a melhoria da qualidade de software e o aumento da produtividade no seu processo de desenvolvimento. A produtividade na construção de software também está relacionada com o aspecto de comunicação que a ferramenta proporcionará, buscando um melhor entendimento do domínio da aplicação tanto da equipe de desenvolvimento como dos usuários.

3.1.1 Ferramentas CASE

Os métodos estruturados a muito adotaram a indústria de ferramentas computadorizadas, projetadas para ajudar no trabalho com estes métodos. Estas ferramentas, chamadas de ferramentas CASE (Computer Aided Software Engineering) têm prometido muito e realizado pouco [Gane'90].

3.1.2 O que é uma ferramenta CASE?

Desde os primeiros dias da escrita de software, que vem sendo dada atenção a necessidade de ferramentas automatizadas para ajudar o desenvolvedor de software. Inicialmente concentrou-se esforços nas ferramentas de suporte a programas como os tradutores, compiladores, montadores, processadores de macros, ligadores e carregadores. Contudo, como os computadores ficaram mais poderosos, e como o software desenvolvido ficou maior e mais complexo, a abrangência das ferramentas de suporte foi expandida. Particularmente com o uso de sistemas "time-sharing" para o desenvolvimento de software, encorajaram o desenvolvimento de editores de programas, "debuggers", analizadores de código, e geradores de relatório [Gane'90].

Como os computadores tornaram-se mais confiáveis e mais populares, a necessidade para a expansão do desenvolvimento de software ficou latente. O desenvolvimento de software veio para ser visto como:

- Uma atividade em larga escala envolvendo um esforço significativo para o estabelecimento dos requisitos, projetar uma solução apropriada, implementar esta

solução, testar esta solução corretamente e documentar as funcionalidades da aplicação final.

- Um grande processo de produção de software requer um maior tempo de vida útil deste software. A implicação disto é que a estrutura do software deve permitir facilmente a adição de novas facilidades. Registros detalhados dos requisitos, dos projetos, da implementação e dos testes do sistema devem ser preservados para permitir a manutenção do software. Adicionalmente, múltiplas versões de todos os produtos produzidos durante o projeto devem ser mantido para facilitar o grupo de desenvolvimento de sistemas de software.
- Um grupo de atividades envolvendo interações entre um número de pessoas durante cada estágio do ciclo de vida de desenvolvimento de uma aplicação. Grupos de pessoas devem ser capazes de cooperar, de uma maneira controlada, e ter uma visão coerente dos estados do projeto.

Esta visão de programação em larga escala resultou em uma maior área de abrangência do suporte a ferramentas sendo desenvolvidas. Contudo, dois fatores importantes influíram no aperfeiçoamento e sofisticação deste tipo de ferramenta:

- Pesquisas na área processos de desenvolvimento de software aumentou o número de métodos de projeto de software (ex: Programação Estruturada de Jackson, o Método de Yourdon) que podiam ser usados como base para o desenvolvimento de software. Estes métodos estavam idealmente apropriados para suportarem ferramentas automatizadas no que elas solicitassem passo a passo, e esta solicitação fosse aderente ao método, tendo notações gráficas associadas a ele, e produzido um grande número de produtos (ex: diagramas, anotações, e documentação) que necessitava ser gravado e mantido.
- Workstations pessoais e computadores pessoais. Estas máquinas tinham relativamente uma grande capacidade de armazenamento, rápidos processadores, e sofisticados monitores gráficos capazes de mostrar tabelas, modelos gráficos, e diagramas.

Nós referenciamos todas as ferramentas que possuem as características acima como ferramentas CASE e elaboramos a seguinte definição:

> *Uma ferramenta CASE é um produto baseado em computador objetivando suportar uma ou mais atividades de um processo de desenvolvimento de software.*

Outros autores têm tentado fazer distinção entre diferentes classes de ferramentas CASE. As diferenças mais comuns são:

- Entre aquelas ferramentas que são *interativas* em sua natureza (como uma ferramenta que suporta um método de projeto) e aquelas que não são (como um compilador). Na forma clássica as primeiras são clamadas de ferramentas CASE, enquanto as últimas são chamadas de ferramentas de desenvolvimento.
- Entre aquelas ferramentas que suportam atividades iniciais no ciclo de vida de um projeto de software (como ferramentas de requisitos e suporte a projeto) e aquelas que são usadas após o ciclo de vida (como compiladores e ferramentas de suporte a teste). Na forma clássica as primeiras são chamadas de ferramentas CASE front-end, e as últimas são chamadas de ferramentas CASE back-end.

Infelizmente, todas estas diferenças são problemáticas. No primeiro caso, é muito difícil dar uma simples e consistente definição de "interativo" que seja expressivo. No segundo caso este é um assunto a respeito de métodos e abordagens em uso (ex: desenvolvimento de software orientado a objetos, ou desenvolvimento de protótipo), disso nossa definição genérica de uma ferramenta CASE.

3.2 Sinergismo de Tecnologias de Software

Uma tecnologia sozinha não pode proporcionar grandes mudanças. Elas devem ser combinadas a outras tecnologias de software. As técnicas orientadas a objeto tornaram-se muito poderosas quando combinadas a outras tecnologias.

Abaixo relação de um conjunto de tecnologias voltadas para o desenvolvimento de software. O sinergismo dessas tecnologias trará uma grande revolução no software [Martin'96].

Tecnologias Voltadas para o Desenvolvimento do Software

- CASE e I-CASE
- Programação Visual
- Geradores de Código
- Repositório
- Metodologias Baseadas em Repositórios
- Engenharia de Informação
- Bancos de Dados Orientados a Objeto

- Linguagem Não-Procedural
- Máquina de Inferência
- Tecnologia Cliente/Servidor
- Bibliotecas de Classes que Maximizam a Reusabilidade
- Análise e Projeto Orientados a Objeto

Abaixo apresentamos um resumo das tecnologias voltadas para o desenvolvimento do software apresentadas acima, segundo [Martin'96].

- **CASE (Engenharia de Software Auxiliada por Computador)** Particularmente importante para o futuro do desenvolvimento do software é o crescimento da indústria CASE. As ferramentas CASE são o equivalente às ferramentas CAD da indústria (Projeto Auxiliado por Computador).

  - As ferramentas CASE empregam representações gráficas na tela ajudando a automação da análise, projeto e geração de software.
- **I-CASE (CASE Integrado)** Refere-se a um conjunto de ferramentas CASE que suportam todas as fases do ciclo de vida, inclusive a completa geração de código, com um único repositório logicamente consistente.
- **Programação Visual** A programação visual permite que os desenvolvedores introduzam, entendam, imaginem, executem e manipulem programas usando notações pictográficas.
- **Geradores de Código** Os geradores de código produzem código sem nenhum erro de sintaxe a partir de projetos, gráficos e especificações de alto nível.
- **Repositório** O repositório é um mecanismo para definir, armazenar e gerenciar as informações sobre uma empresa, seus dados e sistemas.

  O coordenador de repositório aplica métodos aos dados do repositório para garantir que os dados e suas representações CASE tenham consistência e integridade.
- **Metodologias Baseadas em Repositórios** Uma Metodologia Baseada em Repositório é idealizada para tirar proveito integral de um conjunto de ferramentas I_CASE e para maximizar a reusabilidade.
- **Engenharia de Informação** A Engenharia da Informação aplica técnicas de projeto e modelagem integradas à empresa como um todo (ou a uma grande parte da empresa), e não simplesmente a um único sistema.

- **Banco de Dados Orientados a Objetos** Um banco de dados orientado a objeto é projetado para armazenar dados objeto e métodos com técnicas eficientes para o processamento orientado a objeto.
- **Linguagens Não-Procedurais** As Linguagens não-procedurais definem o que é desejado, não como é programado.
- **Máquina de Inferência** Uma máquina de Inferência é o software que faz deduções a partir de fatos e regras, usando técnicas de inferência lógica.
- **Tecnologia Cliente/Servidor** Um Cliente é um módulo de software que solicita uma operação. Um Servidor é um módulo de software que responde a essa solicitação.
- **Bibliotecas de Classes** Uma biblioteca de classes contém implementações reusáveis de tipos de objeto. Seu intento deve ser o de conseguir o máximo grau de reusabilidade no desenvolvimento de software.
- **Análise e Projeto Orientados a Objeto** A análise e projeto orientados a objeto modelam o mundo em termos de objetos que têm propriedades e comportamento, e eventos que disparam operações que mudam o estado dos objetos. Os objetos interagem formalmente com outros objetos.

3.3 Tecnologia CASE

3.3.1 O que é um Ambiente CASE?

A primeira geração de ferramentas CASE teve grande parte do seu desenvolvimento concentrado na automação de tarefas isoladas, tais como a produção da documentação, controle de versões do código fonte, e suporte aos métodos de projeto. Enquanto o sucesso era reconhecido no suporte a tarefas específicas, a necessidade destas "ilhas de automação" serem conectadas vinham claramente sendo reconhecidas por muitos usuários da primeira geração de ferramentas CASE. Por exemplo, um cenário típico de desenvolvimento requer que projetos sejam atentamente relatados em seu código fonte resultante, que eles tenham sua descrição consistente em um conjunto de documentação, e que todos estes produtos estejam sob um controle centralizado de versões. As ferramentas que suportam as tarefas individuais do projeto, codificação, documentação, e controle de versão poderiam ser integrados se elas suportarem este tipo de cenário eficientemente [Martin Fowler'95].

De fato, estas ferramentas são freqüentemente usadas como componentes em um mais elaborado software de desenvolvimento, suportando infra-estrutura que é disponível para a

engenharia de software. Um típico ambiente CASE consiste de um número de ferramentas CASE operando em um hardware comum e numa mesma plataforma de software. Note que existe um número de diferentes classes de usuários de um ambiente CASE. Alguns usuários, como os desenvolvedores de software e gerentes, querem fazer uso de ferramentas CASE para usa-las no desenvolvimento de sistemas de aplicação e monitoração do progresso de um projeto. De outra maneira, integradores de ferramentas são responsáveis pela segurança que as ferramentas operadas em uma plataforma de hardware e software disponibilizam, e o papel do administrador de sistemas é manter e atualizar a própria plataforma de hardware e software.

Note também que desenvolvedores de software, integradores de ferramentas, e administradores de sistemas interagem com múltiplas ferramentas CASE e componentes do ambiente que formam a plataforma de hardware e software do ambiente CASE. São estas interações, entre os diferentes componentes do ambiente CASE e entre usuários e esses componentes, que são o elemento chave de um ambiente CASE. As várias maneiras de abordagem para o gerenciamento, controle, e suporte destas interações fazem a distinção de um ambiente CASE de outro. Nós podemos definir um ambiente CASE pela ênfase e importância destas interações:

> *Um ambiente CASE é uma coleção de ferramentas CASE e outros componentes com uma abordagem de integração que suporta muitas ou todas as interações que ocorrem entre os componentes do ambiente, entre os usuários do ambiente e o próprio ambiente.*

A parte crítica desta definição é que as interações entre os componentes do ambiente são dentro do ambiente. O que distingue um ambiente CASE de uma mistura aleatória de ferramentas CASE é que existe "interface" que é fornecida ao ambiente para facilitar a interação entre estas ferramentas. Esta "interface" pode ser um mecanismo físico como um banco de dados compartilhado, ou um sistema de distribuição de mensagens; uma noção conceitual filosófica de compartilhamento na arquitetura das ferramentas ou semântica comum dos objetos manipulados pelas ferramentas, ou alguma combinação destas coisas.

A área das possíveis maneiras de fornecer uma "interface" que ligue estas ferramentas CASE, inevitavelmente liderará o espectro das abordagens para implementação de um ambiente CASE. Um dos principais pontos que devemos levar em conta é que existem muitas maneiras de construir um ambiente CASE. Enquanto muitas pessoas concentram-se na seleção de ferramentas CASE e componentes para a montagem de um ambiente CASE, eles ignoram a necessidade do suporte a interações entre estes componentes. Centrarmo-nos menos em quais componentes podem ser escolhidos, e mais em como selecionar estes componentes que possam

trabalhar juntos efetivamente. Para saber se uma abordagem escolhida para a interação de componentes é apropriada em um dado contexto irá depender de muitos fatores: as necessidades da empresa em questão, a disponibilidade de recursos, e assim por diante. Uma avaliação detalhada destes fatores e limitações são necessárias para determinar o ambiente CASE mais apropriado para o problema.

3.3.2 Ferramentas Simples e Multi métodos

Uma importante distinção entre as ferramentas é saber se elas são simples ou multi métodos. Uma ferramenta simples é projetada para suportar somente um método. A ferramenta é ligada ao método de projeto, por exemplo "Objectory" (Jacobson) ou Rational Rose (Booch). Eles contrastam com ferramentas multi métodos que fornecem suporte para múltiplos métodos. Tais ferramentas são flexíveis e permitem ao usuário escolher o método com que quer trabalhar antes de começar a modelar. Ferramentas simples obviamente prendem o usuário naquele método e o "orientam" para que ele não fuja do que o projetista da ferramenta definiu. Ferramentas multi métodos podem ser usadas com uma variedade de métodos e a escolha do método não é imposta ao usuário. Ferramentas multi métodos devem possuir um ambiente meta projeto (meta-design) o qual permite que alguém descreva um método para a ferramenta, isto permite que métodos sejam adicionados ou mudados muito mais facilmente que ferramentas simples métodos [Martin Fowler'95].

Contudo ferramentas multi métodos raramente suportarão tão bem um método quanto uma ferramenta simples, por que é mais difícil fornecer suporte a muitos métodos, um único método torna o suporte mais fácil. As ferramentas multi métodos não são flexíveis como se suporia. Na prática, os desenvolvedores acham que eles podem adicionar novos conceitos a um método existente. Isto só é possível com ferramentas multi métodos se o ambiente de meta projeto (meta-design) for aberto e usado pelo pessoal de desenvolvimento. Na prática isto acontece raramente. Deste modo é como se uma ferramenta multi métodos tivesse embutida muitas ferramentas simples método. Para ser verdadeiramente útil uma ferramenta CASE deveria ser facilmente "arrumada" de acordo com os requisitos específicos do lugar em que ela fosse instalada.

Em consideração ao usuário de ferramentas CASE, é essencial que se avalie quais produtos a ferramenta liberará para o pessoal do desenvolvimento. Assim estes produtos podem ser apropriadamente avaliados na comparação com outras maneiras de fazer a mesma coisa.

3.3.3 Desenvolvendo Ferramentas

Desenhar modelos gráficos no papel é tempo consumido e propenso a erros, particularmente porque eles necessitam de revisões freqüentes. Assim ferramentas dedicadas a desenhar podem suportar um nível similar de suporte para engenharia de software assim como as ferramentas CAD fornecem em outras disciplinas [Martin Fowler'95].

A melhor ferramenta para a engenharia de software é um papel branco. É rápido e fácil de desenhar e apagar, visível a várias pessoas, e flexível. Isto encoraja idéias a serem tratadas e descartadas. Nenhuma ferramenta de computador pode se basear nesta facilidade. Contudo, para proporcionar uma documentação apropriada algumas ferramentas de computador são bem recebidas. A primeira possibilidade a ser considerada é a padronização das ferramentas de projeto. Tal qual uma coisa que seja barata e robusta (padrões de ferramentas CASE) e rode numa plataforma desktop padrão. O elemento principal que falta nas ferramentas de projeto é a facilidade de fazer linhas ligadas a retângulos Assim quando um retângulo for movimentado de lugar a linha o acompanha. O tipo de limitação imposta às ferramentas CASE nos desenhos freqüentemente desestimulam seu uso.

3.3.4 Repositório

Além de uma ferramenta de desenho, as ferramentas podem oferecer um repositório: essencialmente um banco de dados conectado a um diagramador. Um banco de dados é capaz de listar todas as classes em ordem alfabética com uma lista de todas as características de uma classe (associação, atributos, operações, etc) com outra classe. Isto pode consolidar modelos expressados em muitos diagramas.

O conteúdo das ligações entre os elementos é o que é importante aqui. Um simples teste é a mudança do nome de uma classe em um modelo, e ver como a ferramenta ajuda na preservação dos dados. Idealmente a ferramenta deveria mudar todas as referencias da classe existentes no repositório, o que freqüentemente não ocorre. Deve haver uma área particular para guardar as referencias de classe dentro de um texto para descrição informal. Normalmente isto não é mudado, e em nenhuma ferramenta é fornecida ajuda de como fazer estas mudanças. Outra área a considerar é a habilidade de seguir ligações entre as diferentes técnicas. Se uma operação chamar um diagrama de interações atribuída a uma classe neste trabalho, isto mudará também? A habilidade de seguir dos desenhos e ligações serem mantidas é particularmente importante com multi métodos que mantém separado os modelos de análise e projeto. Para isto

não ser quebrado, a ferramenta deve fornecer um bom suporte para o modelador para que ele veja os resultados de "uma pequena mudança" feita [Martin Fowler'95].

Outro importante elemento a ser considerado é como estas importantes informações serão acessadas. Ferramentas devem permitir a produção de documentos do projeto com qualidade e com o mínimo de esforço. Novamente isto é uma falha comum, muitas ferramentas podem levar muitos dias de esforço na colocação do contexto do projeto em um formulário apresentável.

O básico aqui é ter um modesto processador de texto. Isto fornece excelentes facilidades na documentação de um projeto. A desvantagem de um processador de texto é a manutenção dos passos com as mudanças nos diagramas, necessitando de tempo e experiência. Contudo este tempo pode ser menor que o tempo levado para "geração" de um documento de projeto a partir de uma ferramenta CASE, e o processador de texto tem mais capacidade e facilidade de pesquisar e trocar todas as referencias para um nome. Outra alternativa pode ser o uso de um banco de dados simples.

3.3.5 Geração de código e execução

Geração de código é provavelmente a característica mais importante de uma ferramenta CASE. Um importante conceito para começar é que a palavra "geração de código" é realmente outro termo para "compilador" e geradores de código devem ser julgados de acordo com os mesmos padrões e com as mesmas considerações.

Uma das primeiras questões a ser respondida é "Quanto de código deve ser gerado?". Muitos geradores de código são somente geradores de interfaces: definição de classes com atributos e operações, não gerando código que possa levar a qualquer processamento. Com isto estas ferramentas são necessariamente para mostrar como as interfaces serão apresentadas. São operações de acesso e atualização gerado para todos os atributos? As operações são simplesmente nomes, ou elas contem uma correta identificação? Operações são geradas somente a partir de listas marcadas numa classe na visão estrutural, ou elas são geradas a partir de um diagrama de estados? [Martin Fowler'95].

A limitação óbvia desta abordagem é que somente as interfaces são fornecidas, o resto do código precisa ser feito. Uma segunda consideração é que com a evolução do projeto, como o modelo e o código serão manuseados. Um gerador de código é de uso limitado se ele não suporta a implementação conseqüente de mudanças no modelo de análise e projeto. Muitas ferramentas CASE assumem que o código só é gerado ao final do projeto. Os projetos reais de análise, projeto e implementação envolve-se com freqüência no modelo em espiral do

desenvolvimento de software[Pressman'95]. Geradores de interface precisam fornecer o fechamento das ligações entre a ferramenta CASE e o ambiente de programação.

Algumas ferramentas suportam engenharia reversa. Isto permite um código final, preferivelmente incluindo código não gerado pela ferramenta CASE, e que seja lido depois pela ferramenta CASE. Desta maneira a ferramenta fornece um grande grau de compreensibilidade do código fonte. Naturalmente tal abordagem demanda muitas técnicas de projeto por isso não será razoável a espera por uma ferramenta para análise da engenharia reversa.

Além disso, um importante passo nos geradores de interfaces é a construção de modelos executáveis. De modo sem qualquer trabalho extra, além do ambiente CASE, seja possível executar o modelo como se ele fosse qualquer outro programa de computador. A vantagem disto é que o modelo pode ser testado diretamente. Isto ajudará no entendimento e na detecção de erros. O programa gerado pode ser levado para compilação ou interpretação, com as vantagens e desvantagens de cada um. Note que compilação implica que o código gerado, não deve ser alterado nem adicionado a outro programa.

Produção de código para compilação, para ser digna do nome, deve dizer que a ferramenta pode produzir um código com qualidade de produção. Este código deve ser apropriadamente controlado pela ferramenta CASE, de modo que a ferramenta CASE é o ambiente de programação e pode ser usada para manter o código. Esta é a diferença entre compilação de código em um sistema liberado e compilação de código em uma arquitetura orientada a objetos. O último busca permitir aos desenvolvedores usar as classes geradas como componentes em outros sistemas, o qual pode ou não ser gerado pela mesma ferramenta. Uma informação adicional é para saber em que grau os desenvolvedores poderiam influenciar o estilo do código gerado. Uma ferramenta como a [Ptech] gera código C++, mas o estilo é muito particular da ferramenta. Em oposição a isto está a ferramenta [Bridgepoint], que permite aos desenvolvedores fornecerem seus próprios templates (padrões e modelos de programação) para o gerador de código. Isto permite ao gerador de código seguir exatamente o estilo do usuário. Isto também é necessário para a ligação efetiva com outras ferramentas, ou suportar linguagens adicionais. Muitas ferramentas (incluindo Ptech) estão aderindo ao uso de template na geração de código [Martin Fowler'95].

Prototipagem de sistemas permite a ferramenta CASE simular toda a execução de um sistema. Isto deve incluir as interfaces do usuário e pode incluir ligações para sistemas remotos e banco de dados. Isto é provavelmente a característica mais útil para qualquer ferramenta CASE, suportando análise e prototipagem totalmente integradas. Simulações permitem que o modelo seja usado para estudos. Objetos aleatórios podem ser gerados de

acordo com probabilidades estatísticas e o modelo roda sobre um período de tempo gerando resultados gráficos. Algumas simulações são particularmente valiosas para a engenharia empresarial.

4. Metodologia FADO

4.1 Introdução

A metodologia FADO está inserida em um contexto maior chamado DYNAMO (DYNAmic Management of Objects) [Schiel'92] que será sucintamente descrito.

O ambiente DYNAMO é um projeto em desenvolvimento pelo grupo de Sistema de Informação e Bancos de Dados do DSC/UFPb, para criação de um Ambiente de Desenvolvimento de Sistemas (ADS) com características temporais e ativas. Ele inclui duas metodologias de projetos de sistemas de informação: POKER (Petri-Net Oriented Knowledge Engineering Research) [Schiel'96] e FADO (Ferramenta de Análise e Desenvolvimento Orientada a Objetos) [Furtado'93] .

O DYNAMO objetiva apoiar desenvolvedores de software na automatização de seus processos de desenvolvimento através do monitoramento de projetos e aperfeiçoamento de processos.

A Figura 4-1, mostra a interação de suas ferramentas e de seus usuários. O acompanhamento da geração dos seus produtos nas diferentes fases do projeto apoia a gerência na detecção de falhas e qualidade obtidas, portanto, identificando possíveis desvios que possam vir a ocorrer.

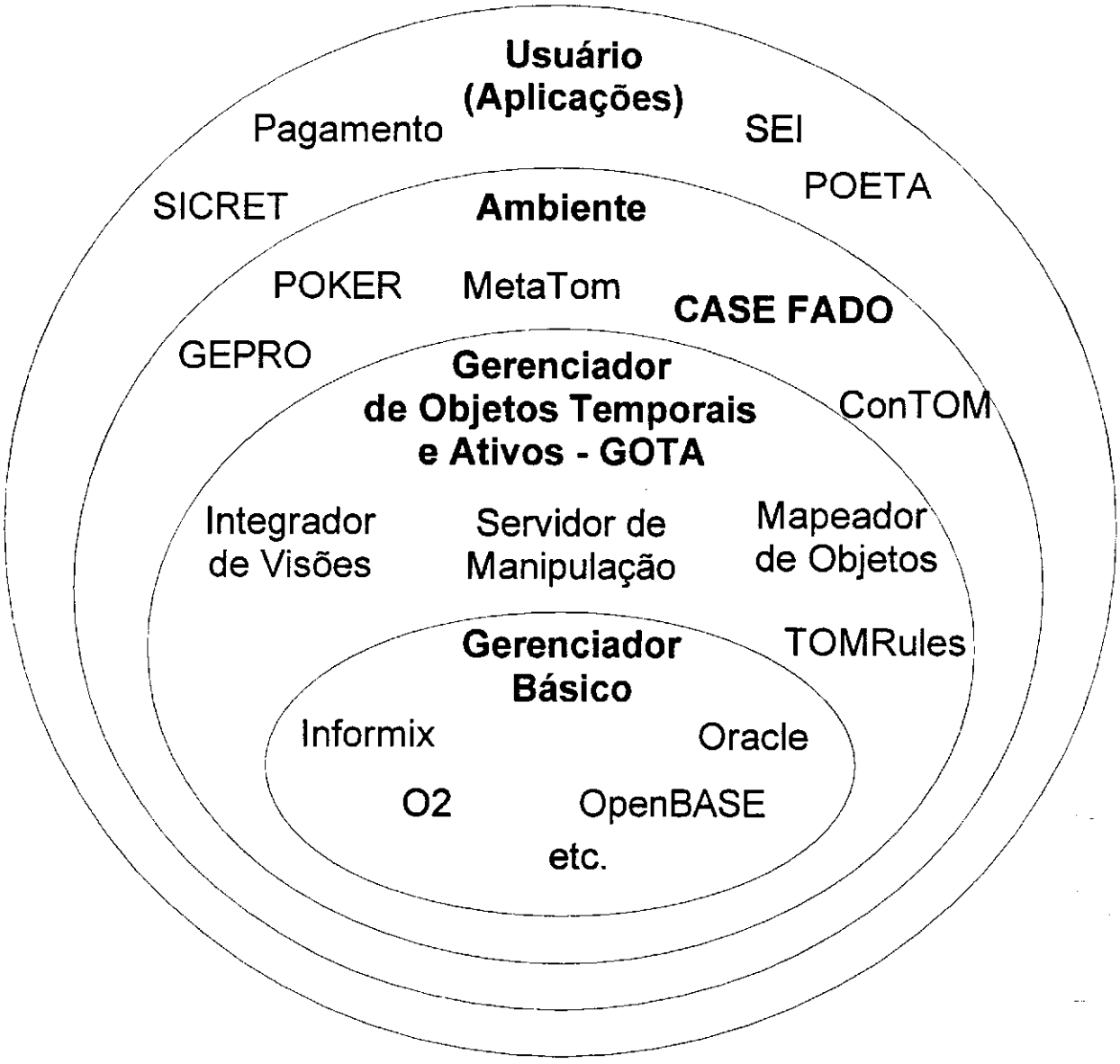


Figura 4-1 - Ambiente DYNAMO

4.2 Modelo Orientado a Objeto Temporal -TOM

TOM (Temporal Object Model) é um modelo de dados orientado a objetos e temporal [Schiel90]. O modelo baseia-se em dois aspectos: estático e dinâmico. O aspecto estático, cuida da estrutura dos dados e é baseado nos conceitos de classe, relacionamento, abstrações (agregação, generalização e agrupamento) e tempo. A implementação deste aspecto representará o esquema de banco de dados da aplicação. O aspecto dinâmico preocupa-se com o comportamento da aplicação em conseqüência da ocorrência de eventos acarretando mudanças de estado dos objetos. Este aspecto baseia-se nos conceitos de método, evento, *trigger* e regra.

4.2.1 Aspectos Estáticos

4.2.1.1 Classe

No mundo real toda e qualquer entidade pode ser representada por um objeto, instância de uma classe no modelo. Para uma classe são definidos relacionamentos, abstrações e métodos que se aplicam a todas as instâncias da classe.

Classes podem ser primitivas, não primitivas e temporais, de acordo com suas instâncias. Classes primitivas são aquelas cujas instâncias são identificadas por tipos simples de dados (inteiro, cadeia de caracteres, etc); as não primitivas correspondem aos tipos abstratos nos modelos semânticos e suas instâncias são identificadas por um ou mais relacionamentos chave (ex: empregado); as temporais são aquelas cujas instâncias, quando removidas da classe, permanecem no banco de dados com uma indicação de quanto tempo pertenceram àquela classe.

David em [David'92] considera uma classe como um sistema composto de duas partes:

1. Descrição da classe (intenção): contendo o nome da classe, uma lista de relacionamentos, uma lista de métodos, sua posição na hierarquia de classe, suas subclasses e superclasses (metaclasses), e parâmetros de tempo.
2. Conjunto de instâncias (extensão): composto de um conjunto de objetos em que os relacionamentos descritos na classe assumem valores concretos para cada objeto (o seu estado), e métodos da classe.

4.2.1.2 Metaclasse

O modelo TOM possui dois níveis de abstração, o metanível e o nível de aplicação. O metanível contém várias metaclasses que descrevem a sintaxe (estrutura) e a semântica (comportamento) do modelo de dados e são independentes de aplicações. Neste nível o Administrador de Modelos (AM) pode criar ou modificar o modelo de dados, ditando novos conceitos e abstrações que serão representados como metaclasses. Neste nível é possível alterar o próprio modelo de dados, por isso ele é considerado aberto [David92]. No modelo TOM a metaclasse TOM_Class é a principal metaclasse pré-definida. Ela define a estrutura das classes do modelo e toda classe da aplicação será instância de TOM_Class.

O nível de aplicação pode ser usado por um projetista para criar classes como instâncias ou subclasses das metaclasses do metanível, que foram definidas pelo AM, para representar conceitos do modelo em uma aplicação específica.

A Figura 4.2 mostra as possíveis relações de abstração entre classes e metaclasses.

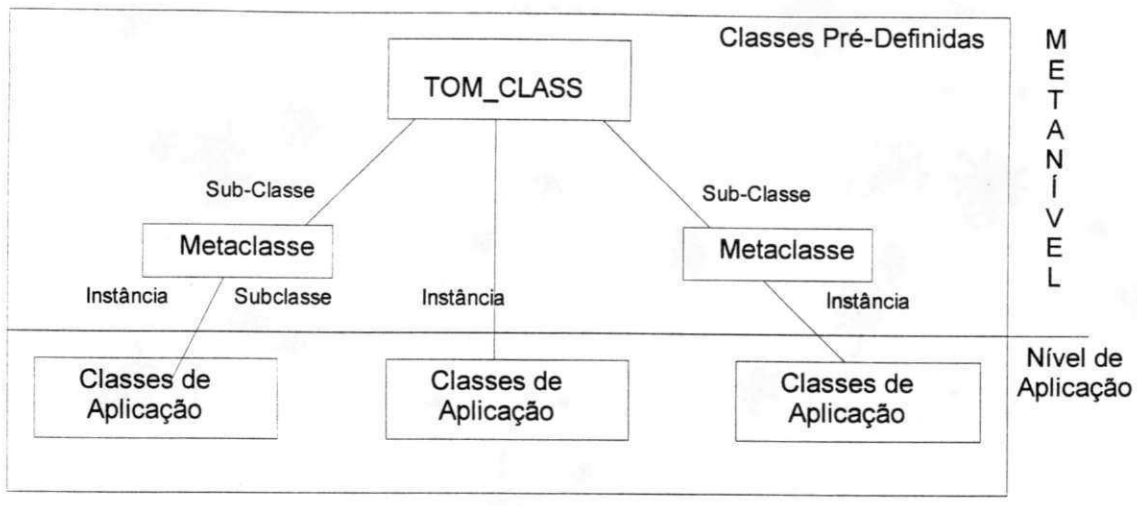


Figura 4-2 Níveis de abstração do TOM

4.2.1.3 Relacionamentos

Um relacionamento representa a informação sobre uma associação entre dois objetos. Cada relacionamento no modelo TOM possui uma cardinalidade, com uma restrição de integridade, expressa por valores (min, max) que determinam o mínimo e o máximo de instâncias associadas pelo relacionamento. Existem os seguintes tipos de relacionamento no modelo TOM:

1. **Relacionamento de Instância** - relaciona instâncias de uma classe com instâncias de outra classe.
2. **Relacionamento de Classe** - relaciona a classe como um todo a uma instância de outra classe.
3. **Relacionamento Temporal** - assim como os objetos, também os relacionamentos podem ser temporais, o que significa que relacionamentos passados permanecem no banco de dados com o seu tempo de validade.

4.2.1.4 Abstrações

As classes no modelo TOM suportam todos os mecanismos mais conhecidos de abstrações dos modelos semânticos. Essas abstrações também são chamados de relacionamentos hierárquicos entre classes, pois os detalhes das classes inferiores são suprimidos na formação de classes mais gerais, sem perder as propriedades comuns às classes relacionadas. Esses relacionamentos hierárquicos representam as abstrações e podem ser usados recursivamente na construção de objetos complexos. Dividem-se em generalização/especialização, agregação e agrupamento.

Generalização/Especialização

Generalização é o processo de suprimir as diferenças entre algumas classes, identificando suas propriedades comuns e definindo uma superclasse como generalização. Especialização é o processo inverso, e consiste de selecionar de uma classe de objetos uma subclasse. É identificada pelo predicado é-um(A,B), significando que A é uma subclasse de B.

Agregação

A construção de objetos a partir da composição de outros objetos chama-se agregação. É representada pelo predicado é-parte(A,B) indicando que A é componente da classe agregada B.

Agrupamento

Agrupamento é a forma de abstração que possibilita a definição de objetos a partir de conjuntos de instâncias de uma classe com instancias de outra classe. É representado pelo predicado é-elemento(A,B), onde as instâncias de B são conjuntos de instâncias de A . Por exemplo, grupos de objetos da classe árvore formam instâncias da classe floresta.

Herança

Na generalização, a herança é automática pois toda instância de uma subclasse também pertence à superclasse e portanto herda as propriedades definidas neste nível. Na agregação e agrupamento, a herança é seletiva, para cada relacionamento e método, pode haver herança direta ou herança computada.

Na herança direta os relacionamentos e métodos podem ser herdados pelos objetos de nível inferior, sem modificações. Como exemplo, podemos citar a **classe automóvel**, que possui o relacionamento pertence-a e o método muda-proprietário que serão herdados pelas classes componentes.

Na herança computada, algumas propriedades podem ser herdadas, através de uma computação. Exemplificando temos o relacionamento peso, que é a soma dos pesos dos componentes.

Tempo

Um dos aspectos fundamentais do modelo TOM é a modelagem do tempo. O tempo é modelado através das classes temporais, dos relacionamentos temporais e dos relacionamentos pré-pós [David'92, Furtado'93, Schiel'91]. Nas Classes e relacionamentos temporais, um objeto ou relacionamento, mesmo após não ser mais válido, é mantido no banco de dados para

pesquisas históricas. No relacionamento pré-pós um objeto removido da pré-classe automaticamente será inserido na pós-classe.

4.2.2 Aspectos Dinâmicos

4.2.2.1 Método

Método é um procedimento de mudança de estado dos objetos de uma classe, que se reflete na alteração dos seus relacionamentos e envio de mensagens a outros objetos. Os métodos são acionados por mensagens que solicitam sua execução.

Os métodos podem ser de instância ou de classe. Um método de instância é acionado com uma mensagem enviada a uma instância da classe. No método de classe a mensagem é enviada a uma classe de objetos.

4.2.2.2 Evento

Evento é um sinal ou condição que, ao ocorrer, pode vir a mudar o estado do banco de dados. No sistema de regras denominado TOMRules [Carvalho93], eventos foram definidos como objetos pertencentes à metaclasse Evento. Os objetos da classe Evento possuem: nome, a situação que descreve quando ele ocorrerá e o atributo ativo, que determina se o evento está ativo ou não.

No modelo TOM, existem três tipos de evento. Esses tipos de evento são descritos a seguir:

1. **Evento Externo** - ocorre com a chegada de uma mensagem do mundo real (usuário).
2. **Evento Temporal** - ocorre em um dado instante do tempo. Este evento possui os atributos **Quando**, **Atraso** e **Tipo**. O atributo **Quando** diz o ponto inicial do intervalo de ocorrência do evento, isto é, quanto ele deve iniciar; o atributo **Atraso** especifica a duração de validade do evento, durante quanto tempo ele ficará ativo; e o atributo **Tipo** indica se o tipo do evento é discreto ou contínuo. O tipo discreto é aquele evento ativado e executado em uma fração de tempo especificada. O tipo contínuo é aquele que fica ativado por todo um período de tempo especificado.
3. **Evento Interno** - é o evento induzido por uma operação (método) ou um estado do banco de dados.

4.2.2.3 *Trigger*

Trigger é a forma de execução de uma ou mais ações em conseqüência da ocorrência de eventos, sob certas condições pré-estabelecidas. Eles podem também executar atualizações para manter a consistência do banco de dados ou inibir a execução de operações.

4.2.2.4 Regra

É a associação entre Eventos e *Triggers*, determinando quais Eventos irão ativar quais *Triggers*. São as regras que vão direcionar, dar rota aos Eventos e *Triggers*, isto é, reger o comportamento da aplicação. Na Figura 2-9 foram ilustrados a estrutura e o mecanismo da execução de regras. Após a definição das regras, o atributo regra da classe Evento deve ser preenchido.

Além do evento externo e do evento temporal, que podem provocar a execução de um método de uma classe, existem duas situações que podem acionar métodos:

1) Relacionamento dinâmico, por exemplo quanto um método m1 (p.ex. CadastraMulta) do objeto A (Multas) manda uma mensagem a um método m2 (IndicaTemMulta) de um objeto B (Automóvel) (como um ***call***). A Figura 4-3 mostra um exemplo de relacionamento dinâmico. Neste caso, a execução de 'm2' faz parte da execução de 'm1'.

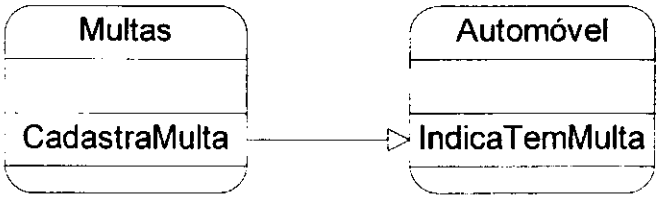


Figura 4-3 Exemplo de relacionamento dinâmico

2) Evento "operação-banco-de-dados", por exemplo quando um método executado ativa um evento que dispara o *trigger* correspondente. Neste caso, o método original não toma conhecimento do *trigger*, como mostra a Figura 4-4.

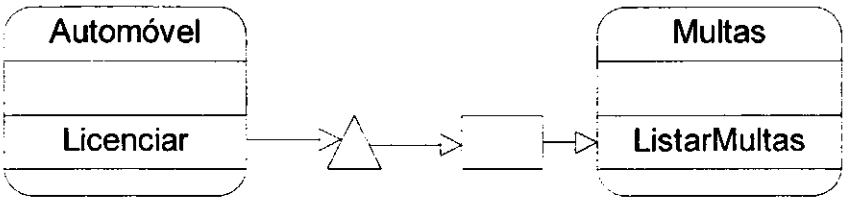


Figura 4-4 Exemplo de evento "operação-banco-de-dados"

4.3 A Metodologia FADO

A metodologia FADO (**F**erramenta de **A**nálise e **D**esenvolvimento **O**rientado a Objetos) [Furtado93] é uma metodologia de auxilio ao projeto de banco de dados temporais orientados a objetos, modelando o domínio da aplicação. A metodologia apresentada aqui, representa uma modificação da versão original de [Furtado'93]. Foi retirado todas as tabelas usadas na versão

original (TAO – Tabela de Objetos Ativos), simplificação de relatórios, modificações e redução no número de processos da metodologia (Figura 4-6) e reforma das fases da metodologia (Figura 4-7), deixando-a mais flexível e fácil de usar. A metodologia é composta de:

Diagramas

Diagrama Estático, Diagrama Contextual, Diagrama Dinâmico, Diagrama de Transição de Estados.

Formulários

Classe, Método, Relacionamento, Abstração, Evento, *Trigger*, Regra, Condição, Generalização, Agregação, Agrupamento e Herança.

Para mostrar o processo de se desenvolver uma aplicação utilizando a metodologia FADO, descreveremos os diagramas e formulários que compõem a metodologia tomando como exemplo o projeto veículo.

4.3.1 Diagrama Estático

Representa o esquema estrutural de uma aplicação, projetando as estruturas das classes e os relacionamentos entre elas. É a partir deste diagrama que montaremos o banco de dados da aplicação. É um diagrama que mostra as hierarquias de classes e os relacionamentos estáticos entre elas. Os tipos de relacionamentos representados são de instâncias, de classes e de abstrações. Os tipos de abstrações desta metodologia podem ser de generalização, agregação e agrupamento conforme o modelo de dados TOM. A Figura 4.5, mostra o diagrama estático de uma aplicação usando a CASE FADO. Para uma aplicação complexa podem ser criados vários diagramas estáticos.

7. Referências Bibliográficas

Baptista A., "Uma Metodologia de Análise e Projeto de Sistemas Orientada a Objetos". Dissertação de Mestrado. UFPE Departamento de Informática. 1992.

Basili, V. and B. Perricone. "Software Erros and Complexity: An Empirical Investigation", Communication of the ACM, vol. 21, january. 1984.

Bertran Meyer. "Object-Oriented Software Construction". Prentice Hall. 1988.

Booch, G., "Object-Oriented Analysis and Design with Aplications. Reading, Mass.: Addison-Wesley, 1994.

Booch, Grady. Architectural Patterns for a competitive advantage. Rational Software Corporation. 1998.

Booch, Grady. Object-oriented development. IEEE Transaction on Software Engineering SE-12, 2, pp. 211-221. fev.1986.

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

BridgePoint. www.bridge-point.com

Brooks. Frederick. "The Mythical Man-Month", Reading: Addison-Wesley Publishing Company, Inc., 1985

Carvalho, A., "Tom Rules: Um monitor de Eventos, Regras e Gatilhos em um Ambiente Orientado a Objetos". Dissertação de Mestrado. UFPB/COPIN. 1993.

Coleman, Derek, Desenvolvimento Orientado a Objetos – O Método Fusion. Editora Campus. 1994.

Computer Design, "CASE Makes Strikes Toward Automated Software Development", 1987.

Constantine, L. L., "Control of Sequence and Parallelism in Modular Programs, In: AFIPS Proceedings of the 1968 Spring Joint Computer Conference, v. 32, 1968. p. 409 es.

Cook, S. e Daniels, J. Designing Object Systems. Englewood Cliffs, N.J.: Prentice-Hall, 1994.

Cox, Brad J. Object-Oriented Programming. Reading, Massachusetts:Addison-Wesley, 1986

Dahl, Ole-Johan e Nygaard, Kristen. "SIMULA – Na Algol-based Simulation Language", Communications of the ACM, 9:9, setembro de 1966, pp. 671-678.

David Harel, "Statecharts: a visual formalism for complex systems", Science of Computer programming, 8, 1987, pp. 231-274.

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

Dilvan Vitor dos Santos. Transformação de Esquemas de Objetos Complexos para um Gerenciador Relacional Extendido, considerando o padrão ODMG. Universidade Federal da Paraíba - Campina Grande, 1998.

Fernandes, Sônia L., "ComTOM – Um sistema de Consultas Gráficas a um Banco de Dados Orientado a Objetos Temporal". Dissertação de mestrado, UFPB, Campina Grande, 1995.

Fernandes, Sônia L., "Um Ambiente Gráfico de Consultas a um Banco de Dados Temporal Orientado a Objetos, em Anais do IX Simpósio Brasileiro de Banco de Dados, São Carlos, setembro 1994, p. 1-15.

Fisher, Alan S., "CASE – Utilização de ferramentas para Desenvolvimento de Software, Editora Campus Ltda, 1990.

Furtado, Maria Elizabeth S., "Uma Metodologia para Projeto de Banco de Dados Temporal Orientado a Objetos", Dissertação de Mestrado, UFPB, Campina Grande, 1993.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J., Design Patterns: elements of reusable object-oriented software. AddisonWesley, MA, 1995.

Gane, Chris. CASE, o Relatório Gane. Livros Técnicos e Científicos Editora Ltda. 1990.

Goldberg, A., e Robson D. Smaltalk-80: The Language. Reading, Mass.: Addison-Weslay,1989.

Graham, I. Migrations using SOMA: a semantically rich method of object-oriented analysis, Journal of Object Oriented Programming, 5,9, 1993, pp. 31-42.

IBM. www.ibm.com

Jackson, M. A., "System Development". Prentice Hall. 1975.

Jacky, Jonathan; Kalet, Ira. Na object-oriented programming in a time series analysis system. OOPSLA'86 como ACM SIGPLAN 21, 11 (nov. 1986), pp. 368-376.

Jacobson. Ivar, Christerson, M., Jonsson, P., e Overgaard, G., Object-Oriented Software Engineering. Reading, Mass.: Addison-Wesley, 1992.

Jones, Gregory W., "Software Engineering. John Wiley & Sons. New York. 1990.

Journal of Object-Oriented Programming.

Kay, A., The Reactive Engine. University of Utah, Department of Computer Science, ago. 1969.

Kelly, J. C., J. Sherif, J. Hops. "Na Analyses of Defect Densities Found During Software Inspections". J. of Systems Software, vol. 17, 1992.

Márcia Jacynta Nunes Rodrigues, dissertação de mestrado – Uma Ferramenta para Modelagem de Sistemas de Informação com Suporte a Diferentes Modelos de Dados, UFPE. Janeiro/97.

Martin Fowler. A Comparison of Object-Oriented Analysis and Design Methods. OOPSLA'95. 1995.

Martin, James. Engenharia da Informação. Editora Campus. 1991.

Martin, James: e Odell, James J., Análise e Projeto Orientados a Objetos. Makron Books. 1996.

Mealy, G. H., A Method of Synthesizing Sequential Circuits. Bell System Technical Journal, v. 34, 1955, p.1045-1079.

MetaDesign 4.0. Meta Software Corporation. 125 Cambridge Park Drive. Cambridge. MA 01240 USA.

Missikoff, M., R. Pizzicannella, "A Visual Approach to OO Analysis Based on Abstract Diagrams, SIGCHI,Vol. 28, Number 3, July, 1996.

Moore, E. F., 'Gedanken-Experiments on Sequential Machines'. Automata Studies, Annals of mathematical Studies, n° 34, 1956, p. 129-53.

Odell, James J., "Specifying requeriments using rules". Journal of Object Oriented Programming. April 1993 – in press. 1993.

Page-Jones, Meilir. O Que Todo Programador Deveria Saber Sobre Projeto Orientado a Objeto. Makron Books, 1997.

Parnas, D., 'Information Distributing Aspects of Design Methodology. Proceedings of the 1971 IFIP Congress. Booklet TA-3. Amsterdã: North-Holland, 1972.

Penedo, Maria H. e William E. Riddle. Guest Editor's introduction to Software engineering environment architectures. IEEE Transactions on Software Engineering, 14(96):689-695, June 1988.

Pressman, Roger S., Engenharia de Software. Makron Books do Brasil, 1995.

Ptech. www.ptechinc.com

Rational Software Corporation. www.rational.com

Revista Developers Magazine, número 12 - agosto/1997, p. 20.

Richards, M., e Whitby-Strevens, C., 'BCPL - The language and Its Compiler. Cambridge. Inglaterra: Cambridge University press, 1980.

Rumabaugh, James; Blaha, Michael; Premerlani, William; Eddy, Frederick; Loresen, Willian. Modelagem e Projetos Baseados em Objetos. Editora Campus, 1994.

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

Schiel, Ulrich, Mistrik, L.. "Using Object-Oriented Analysis and Design for integrated Systems". Procedure International Conference on Systems integration, IESI, New Jersey, 1920.

Schiel, Ulrich. "Vodak Version Model (VVM)", Working Paper, GMD/IPSI, Darmstadt/Germany, 1989.

Schiel, Ulrich. Na Open Environment for Object with Time and Versioning". Procedure EastEurope, Bratislava. 1991.

SEI/CMU, "An Investigation into the State of the Practice of CASE tool Integration, Technical Report, 1995.

Shlaer, Sally and Mellor, Stephen J., Análise de Sistemas Orientada para Objetos. Makron Books. 1990.

Stevens, W., Myers, G. e L. Constantine. "Strutured Design", IBM System Journal, Vol. 13, Nº 2, maio 1974.

Stroustrup, B., The C++ Programming Language. Reading, Mass.: Addison-Wesley, 1991.

Torres, José Belo. Uma ferramenta de Gerência de Projeto - GERPRO. Universidade Federal da Paraíba - Campina Grande, 1996.

Wirfs-Brock, R., Wilkerson, B. e Wiener, L., Designing Object-Oriented Software. Prentice Hall International, Englewood Cliffs, NJ, 1990.

Yourdon, E. e Constantine, L. L., Structured Design. 1ª edition. New York: Yourdon Press, 1975. 2ª edition: Englewood Cliffs, N.J.: Prentice-Hall, 1979.

Apêndice

8. Apêndice

Estudo de caso usando CASE FADO

Introdução

Neste capítulo a ferramenta CASE FADO será utilizada na apresentação dos resultados no desenvolvimento de uma aplicação. Mostrando seus diagramas e relatórios, enfim, como um projeto pode ser desenvolvido usando-se uma ferramenta de auxilio baseada em computador. Na verdade mostraremos o resultado em forma de listagem da documentação gerada pela ferramenta.

As páginas seguintes apresentam todos os relatórios gerados pela ferramenta CASE FADO, este relatórios são o resultado de todos os exemplos e modelos usados no corpo da Dissertação. Para imprimir os resultados do projeto deve-se ir na barra de menu opção arquivo e escolher imprimir, irá aparecer a tela da figura 6-1. Onde pode ser escolhido a maneira de imprimir o projeto, se total ou parcial ficando a cargo do usuário a escolha.

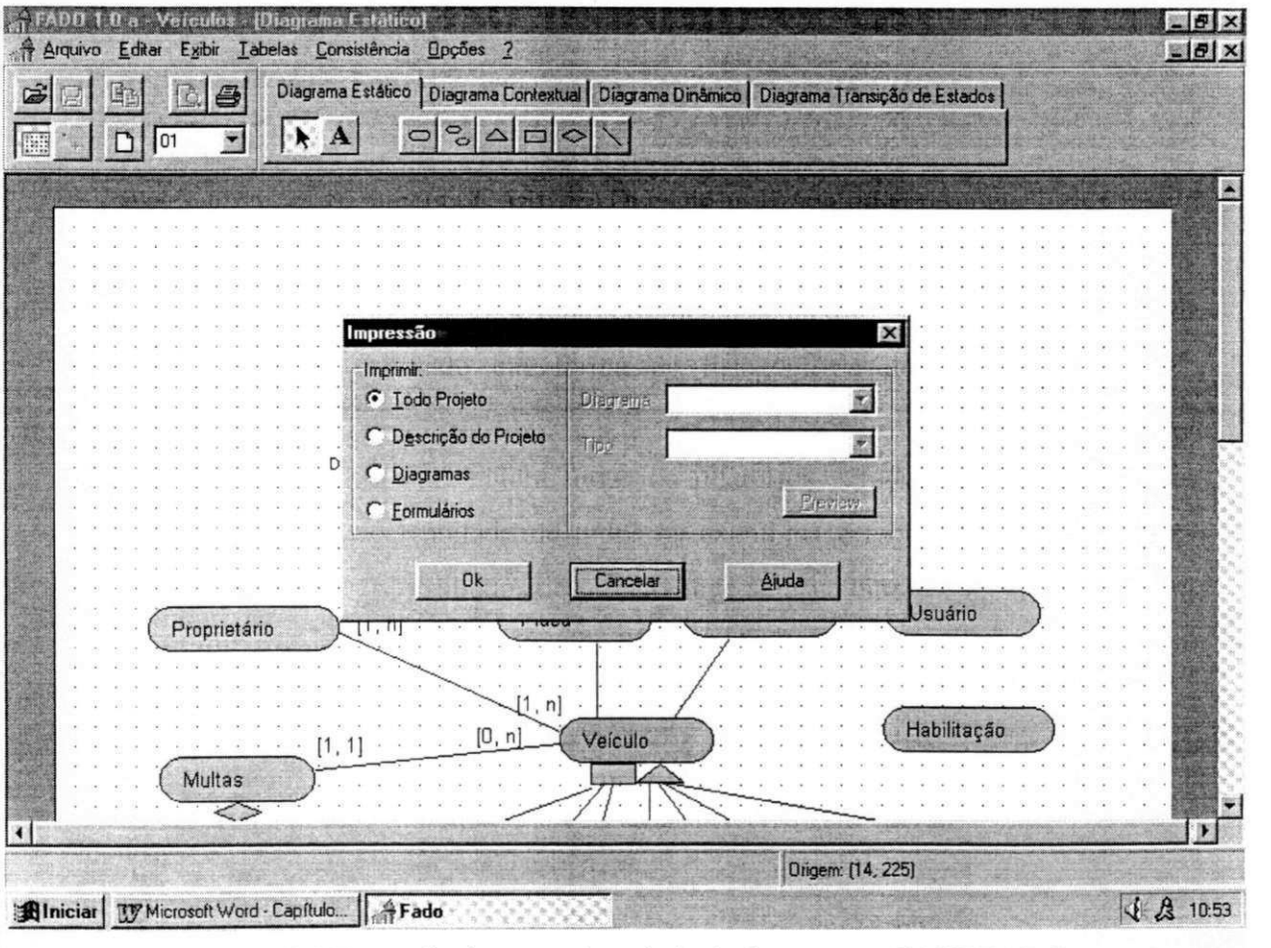


Figura 8-1 Exemplo da opção imprimir da ferramenta CASE FADO

Diagramas

Diagrama Estático

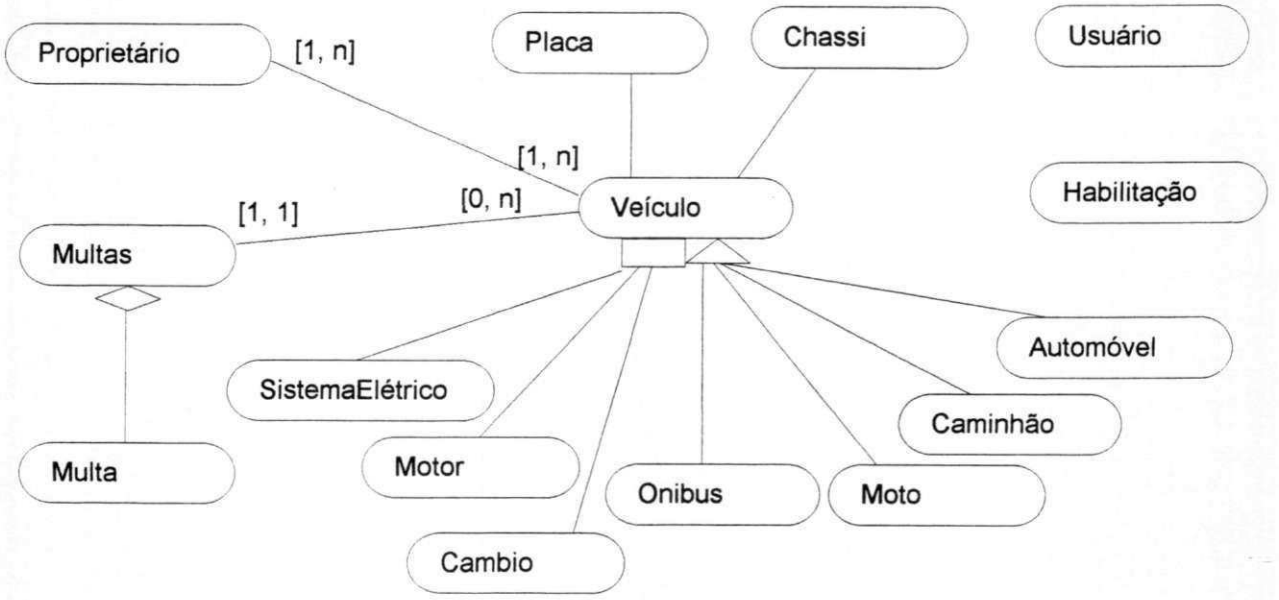


Diagrama Contextual

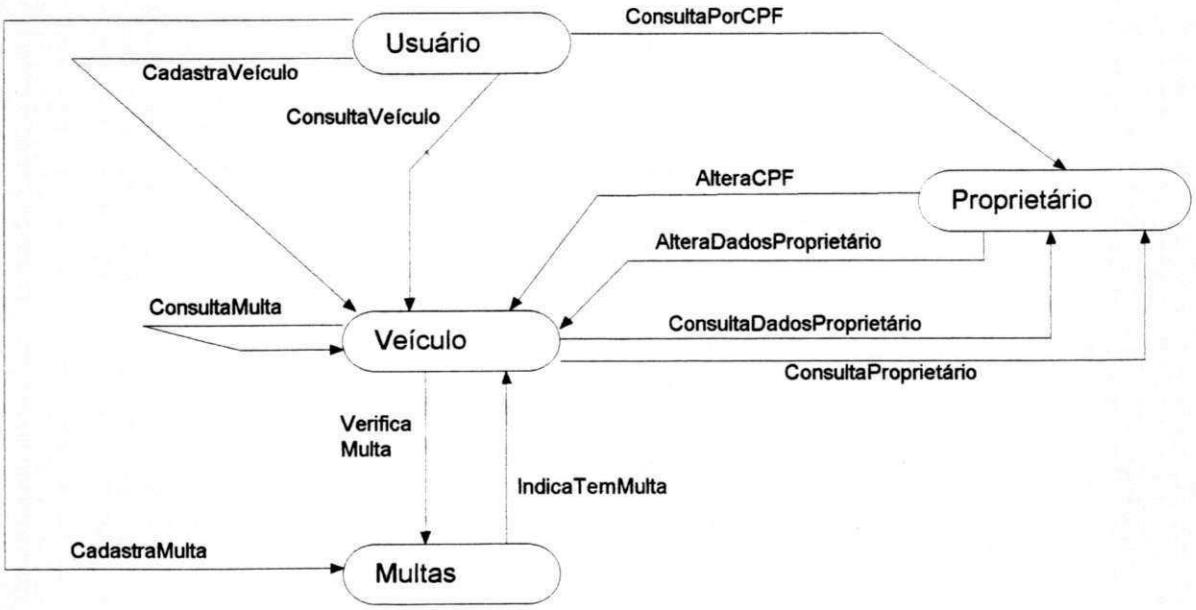


Diagrama Dinâmico

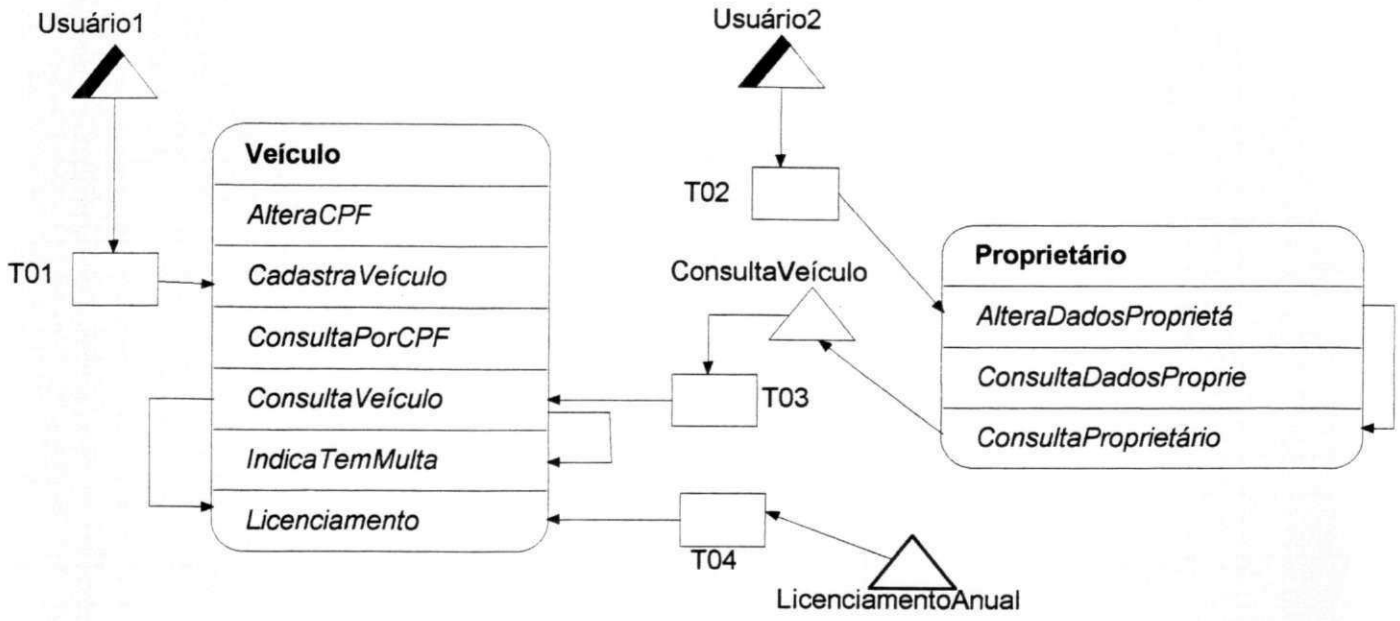
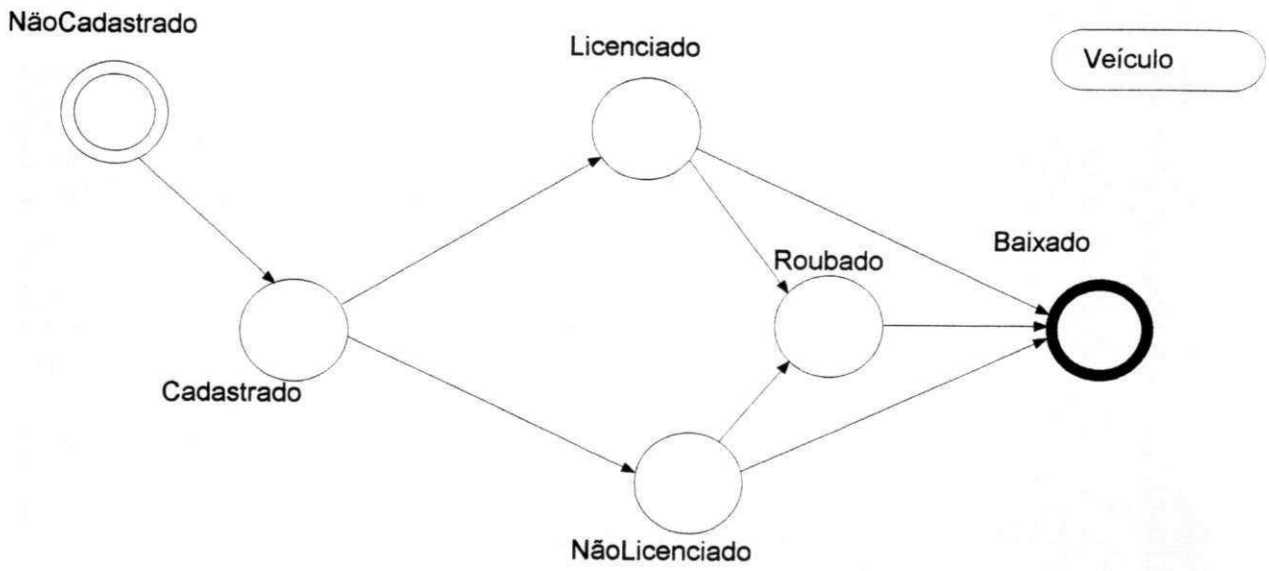
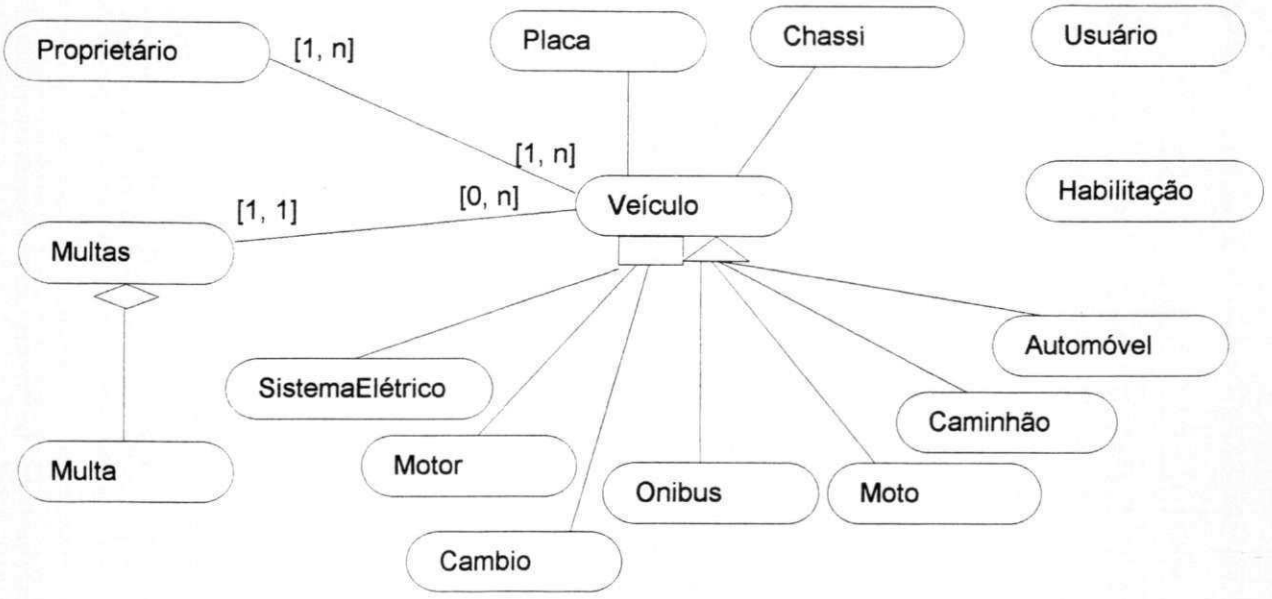


Diagrama de Transição de Estados




Proprietário
[1, n]
Placa
Chassi
Usuário
[1, n]
[1, 1]
[0, n]
Veículo
Habilitação
Multas
Automóvel
SistemaElétrico
Caminhão
Multa
Motor
Onibus
Moto
Cambio

Formulários

Classe	Proprietário
Descrição	Esta classe é responsável por todos os dados do proprietário como por exemplo: Nome, Enderêço, CPF, Identidade e Telefone.
Versão de Domínio	Não
Tipo	
Tamanho	
Temporal	Não
Tipo	Implícito
Início	
Fim	
Metaclasse	Sim

Classe	Veículo
Descrição	Classe responsável pelo cadastramento e manutenção dos dados referentes aos veículos, informando suas características, e dados necessários a sua identificação física e lógica.
Versão de Domínio	Não
Tipo	
Tamanho	
Temporal	Não
Tipo	Implícito
Início	
Fim	
Metaclasse	Não

Classe	Usuário
Descrição	
Versão de Domínio	Não
Tipo	
Tamanho	
Temporal	Não
Tipo	Implícito
Início	
Fim	
Metaclasse	Sim

Classe	Multas
Descrição	
Versão de Domínio	Não
Tipo	
Tamanho	
Temporal	Não
Tipo	Implícito

Classe	Multa
Descrição	
Versão de Domínio	Não
Tipo	
Tamanho	
Temporal	Não
Tipo	Implícito
Início	
Fim	
Metaclasse	Não

Classe	Chassi
Descrição	
Versão de Domínio	Não
Tipo	
Tamanho	
Temporal	Não
Tipo	Implícito
Início	
Fim	
Metaclasse	Não

Classe	Placa
Descrição	Código identificador do veículo, deve ser única. A lei de formação deve obedecer o código nacional de trânsito, cujo formato é 3 letras e quatro números.
Versão de Domínio	Sim
Tipo	Caracter
Tamanho	7
Temporal	Não
Tipo	Implícito
Início	
Fim	
Metaclasse	Não

Classe	Automóvel
Descrição	
Versão de Domínio	Não
Tipo	
Tamanho	
Temporal	Não
Tipo	Implícito
Início	

Classe	Caminhão
Descrição	
Versão de	
Domínio	Não
Tipo	
Tamanho	
Temporal	Não
Tipo	Implícito
Início	
Fim	
Metaclasse	Não

Classe	Moto
Descrição	
Versão de	
Domínio	Não
Tipo	
Tamanho	
Temporal	Não
Tipo	Implícito
Início	
Fim	
Metaclasse	Não

Classe	Onibus
Descrição	
Versão de	
Domínio	Não
Tipo	
Tamanho	
Temporal	Não
Tipo	Implícito
Início	
Fim	
Metaclasse	Não

Classe	Habilitação
Descrição	
Versão de	
Domínio	Não
Tipo	
Tamanho	
Temporal	Não
Tipo	Implícito
Início	
Fim	
Metaclasse	Sim

Classe	Motor
Descrição	
Versão de	
Domínio	Não
Tipo	
Tamanho	
Temporal	Não
Tipo	Implícito
Início	
Fim	
Metaclasse	Não

Classe	Cambio
Descrição	
Versão de	
Domínio	Não
Tipo	
Tamanho	
Temporal	Não
Tipo	Implícito
Início	
Fim	
Metaclasse	Não

Classe	SistemaElétrico
Descrição	
Versão de	
Domínio	Não
Tipo	
Tamanho	
Temporal	Não
Tipo	Implícito
Início	
Fim	
Metaclasse	Não

Abstração	Agrupamento01
Tipo	Agrupamento
Super Classe	Multas
Sub-classes	Multa.
Parâmetros	Completo
Tipo Abstração	Todos

Abstração	Agregação01
Tipo	Agregação
Super Classe	Veículo
Sub-classes	SistemaElétrico, Motor, Cambio.
Parâmetros	Exclusivo
Tipo Abstração	Todos

Abstração	Generalização01
Tipo	Generalização
Super Classe	Veículo
Sub-classes	Automóvel, Caminhão, Moto, Onibus.
Parâmetros	Completo
Tipo Abstração	Explícito

Relacionamento	Aplicadas-ao
Classe Origem	Multas
Classe Destino	Veículo
Tipo	Classe
Relacionamento Inverso	Infrações
Cardinalidade	
Min	0
Max	n
Cardinalidade Inversa	
Min	1
Max	1
Qnt. do Valor Anterior	0
Características do Tempo	

Relacionamento	ÉProprietário
Classe Origem	Proprietário
Classe Destino	Veículo
Tipo	Instância
Relacionamento Inverso	TemDono
Cardinalidade	
Min	1
Max	n
Cardinalidade Inversa	
Min	1
Max	n
Qnt. do Valor Anterior	0
Características do Tempo	

Relacionamento	NumChassi
Classe Origem	Veículo
Classe Destino	Chassi
Tipo	Instância
Relacionamento Inverso	
Cardinalidade	
Min	
Max	
Cardinalidade Inversa	
Min	
Max	
Qnt. do Valor Anterior	0
Características do Tempo	

Relacionamento	PlacaUnica
Classe Origem	Veículo
Classe Destino	Placa
Tipo	Instância
Relacionamento Inverso	
Cardinalidade	
Min	
Max	
Cardinalidade Inversa	
Min	
Max	
Qnt. do Valor Anterior	0
Características do Tempo	

Método	CadastraVeículo
Descrição	QRMemoDescricao
Classe	Veículo
Tipo do Método	Classe
Pré-Condição	0
Pós-Condição	0
Corpo da Ação	QRMemoCorpoAcao
Parâmetros de Entrada	QRMemoParametrosEntrada
Parâmetros de Saída	QRMemoParametrosSaida
Comentário	CadastraVeículo

Método	ConsultaVeículo
Descrição	QRMemoDescricao
Classe	Veículo
Tipo do Método	
Pré-Condição	0
Pós-Condição	0
Corpo da Ação	QRMemoCorpoAcao
Parâmetros de Entrada	QRMemoParametrosEntrada
Parâmetros de Saída	QRMemoParametrosSaida
Comentário	ConsultaVeículo

Método	ConsultaProprietário
Descrição	QRMemoDescricao
Classe	Proprietário
Tipo do Método	Classe
Pré-Condição	0
Pós-Condição	0
Corpo da Ação	QRMemoCorpoAcao
Parâmetros de Entrada	QRMemoParametrosEntrada
Parâmetros de Saída	QRMemoParametrosSaida
Comentário	ConsultaProprietário

Método	AlteraCPF
Descrição	QRMemoDescricao
Classe	Veículo
Tipo do Método	
Pré-Condição	0
Pós-Condição	0
Corpo da Ação	QRMemoCorpoAcao
Parâmetros de Entrada	QRMemoParametrosEntrada
Parâmetros de Saída	QRMemoParametrosSaida
Comentário	AlteraCPF

Método	AlteraDadosProprietá
Descrição	QRMemoDescricao
Classe	Proprietário
Tipo do Método	Classe
Pré-Condição	0
Pós-Condição	0
Corpo da Ação	QRMemoCorpoAcao
Parâmetros de Entrada	QRMemoParametrosEntrada
Parâmetros de Saída	QRMemoParametrosSaida
Comentário	AlteraDadosProprietário

Método	ConsultaDadosProprie
Descrição	QRMemoDescricao
Classe	Proprietário
Tipo do Método	
Pré-Condição	0
Pós-Condição	0
Corpo da Ação	QRMemoCorpoAcao
Parâmetros de Entrada	QRMemoParametrosEntrada
Parâmetros de Saída	QRMemoParametrosSaida
Comentário	ConsultaDadosProprietário

Método	ConsultaPorCPF
Descrição	QRMemoDescricao
Classe	Veículo
Tipo do Método	
Pré-Condição	0
Pós-Condição	0
Corpo da Ação	QRMemoCorpoAcao
Parâmetros de Entrada	QRMemoParametrosEntrada
Parâmetros de Saída	QRMemoParametrosSaida
Comentário	ConsultaPorCPF

Método	CadastraMulta
Descrição	QRMemoDescricao
Classe	Multas
Tipo do Método	
Pré-Condição	0
Pós-Condição	0
Corpo da Ação	QRMemoCorpoAcao
Parâmetros de Entrada	QRMemoParametrosEntrada
Parâmetros de Saída	QRMemoParametrosSaida
Comentário	CadastraMulta

Método	ConsultaMulta
Descrição	QRMemoDescricao
Classe	Multas
Tipo do Método	
Pré-Condição	0
Pós-Condição	0
Corpo da Ação	QRMemoCorpoAcao
Parâmetros de Entrada	QRMemoParametrosEntrada
Parâmetros de Saída	QRMemoParametrosSaida
Comentário	ConsultaMulta

Método	IndicaTemMulta
Descrição	QRMemoDescricao
Classe	Veículo
Tipo do Método	
Pré-Condição	0
Pós-Condição	0
Corpo da Ação	QRMemoCorpoAcao
Parâmetros de Entrada	QRMemoParametrosEntrada
Parâmetros de Saída	QRMemoParametrosSaida
Comentário	IndicaTemMulta

Método	Licenciamento
Descrição	QRMemoDescricao
Classe	Veículo
Tipo do Método	
Pré-Condição	0
Pós-Condição	0
Corpo da Ação	QRMemoCorpoAcao
Parâmetros de Entrada	QRMemoParametrosEntrada
Parâmetros de Saída	QRMemoParametrosSaida

Evento	Usuário1
Descrição	QRMemoDescricao
Tipo do Evento	Externo
Ativo	Sim
Regra	Regra01

Evento	Usuário2
Descrição	QRMemoDescricao
Tipo do Evento	Externo
Ativo	Não
Regra	Regra02

Evento	ConsultaVeículo
Descrição	QRMemoDescricao
Tipo do Evento	Interno
Ativo	Não
Regra	Regra03

Evento	LicenciamentoAnual
Descrição	QRMemoDescricao
Tipo do Evento	Temporal
Ativo	Não
Regra	Regra05
Tipo Evento Temporal	Periódico
Método	Licenciamento
Quando	1
Atraso	1
Periódico	1
Relativo	
Tipo	Contínuo

Trigger	T01
Prioridade	0
Condição	0
Condição Desabilitada	0
Seqüência Execução	
Habilitado	Sim
Métodos	CadastraVeículo

Trigger	T02
Prioridade	0
Condição	0
Condição Desabilitada	0
Seqüência Execução	
Habilitado	Não
Métodos	AlteraDadosProprietá

Trigger	T03
Prioridade	0
Condição	0
Condição Desabilitada	0
Seqüência Execução	
Habilitado	Não
Métodos	ConsultaVeículo

Trigger	T04
Prioridade	0
Condição	0
Condição Desabilitada	0
Seqüência Execução	
Habilitado	Não
Métodos	Licenciamento

Classe: Veículo	**Página: 1**
Estado	NãoCadastrado
Tipo	Inicial
Estado	Licenciado
Tipo	Interno
Estado	NãoLicenciado
Tipo	Interno
Estado	Roubado
Tipo	Interno
Estado	Baixado
Tipo	Final
Estado	Cadastrado
Tipo	Interno

Nome do Projeto	Veículos
Resposáveis	
Nome	Augusto Maia
Nome	Augusto Maia
Descrição	Este projeto é uma simplificação do sistema de veículos.
Data de Início	01/06/96
Data da Previsão para Conclusão	15/11/98
Data da Conclusão	
Tempo Gasto:	___:__

Linguagem TOM/CL gerada pela ferramenta CASE FADO

É gerado alguns arquivos no formato TXT, são eles: BNF_Classe.txt, BNF_Abstração.txt, BNF_Relacionamento.txt, BNF_Método.txt, BNF_Evento.txt e BNF_Trigger.txt, os quais são listados a seguir.

BNF_Classe.txt

Meta classe: Proprietário

Descrição: Esta classe é responsável por todos os dados do proprietário como por exemplo: Nome, Enderêço, CPF, Identidade e Telefone.

Especialização:

Versão:

Dominio: Não

Temporal: Não

Abstração de origem:

Classe: Veículo

Descrição: Classe responsável pelo cadastramento e manutenção dos dados referentes aos veículos, informando suas características, e dados necessários a sua identificação física e lógica.

Especialização:

Versão:

Dominio: Não

Temporal: Não

Abstração de origem:

Classe: Usuário

Descrição:

Especialização:

Versão:

Dominio: Não

Temporal: Não

Abstração de origem:

Classe: Multas
Descrição:
Especialização:
Versão:
Dominio: Não
Temporal: Não
Abstração de origem:

Classe: Multa
Descrição:
Especialização:
Versão:
Dominio: Não
Temporal: Não
Abstração de origem:

Classe: Classi
Descrição:
Especialização:
Versão:
Dominio: Não
Temporal: Não
Abstração de origem:

Classe: Placa
Descrição: Código identificador do veículo, deve ser única. A lei de formação deve obedecer o código nacional de trânsito, cujo formato é 3 letras e quatro números.
Especialização:
Versão:
Dominio: Sim
Tipo de Dominio: Caracter
Tamanho: 7
Temporal: Não
Abstração de origem:

Classe: Automóvel

Descrição:

Especialização:

Versão:

Dominio: Não

Temporal: Não

Abstração de origem:

Classe: Caminhão

Descrição:

Especialização:

Versão:

Dominio: Não

Temporal: Não

Abstração de origem:

Classe: Moto

Descrição:

Especialização:

Versão:

Dominio: Não

Temporal: Não

Abstração de origem:

Classe: Onibus

Descrição:

Especialização:

Versão:

Dominio: Não

Temporal: Não

Abstração de origem:

Classe: Habilitação

Descrição:

Especialização:

Versão:

Dominio: Não

Temporal: Não

Abstração de origem:

Classe: Motor

Descrição:

Especialização:

Versão:

Dominio: Não

Temporal: Não

Abstração de origem:

Classe: Cambio

Descrição:

Especialização:

Versão:

Dominio: Não

Temporal: Não

Abstração de origem:

Classe: SistemaElétrico

Descrição:

Especialização:

Versão:

Dominio: Não

Temporal: Não

Abstração de origem:

BNF_Abstração.txt

Abstracao: Agrupamento01

Tipo: Agrupamento

Parâmetros: Completo

Tipo Abstração: Todos

Abstracao: Agregação01

Tipo: Agregação

Parâmetros: Exclusivo

Tipo Abstração: Todos

Abstracao: Generalização01

Tipo: Generalização

Parâmetros: Completo

Tipo Abstração: Explícito

BNF_Relacionamento.txt

Relacionamento: Aplicadas-ao

Tipo: Classe

Relacionamento Inverso: Infrações

Cardinalidade

    Min: 0

    Max: n

Cardinalidade Inversa

    Min: 1

    Max: 1

Quantidade do Valor Anterior: 0

Característica de Tempo:

Relacionamento: ÉProprietário

Tipo: Classe

Relacionamento Inverso: TemDono

Cardinalidade

    Min: 1

    Max: n

Cardinalidade Inversa

Min: 1

Max: n

Quantidade do Valor Anterior: 0

Característica de Tempo:

Relacionamento: Relacionamento04

Tipo:

Relacionamento Inverso:

Cardinalidade

Min:

Max:

Cardinalidade Inversa

Min:

Max:

Quantidade do Valor Anterior: 0

Característica de Tempo:

Relacionamento: Relacionamento05

Tipo:

Relacionamento Inverso:

Cardinalidade

Min:

Max:

Cardinalidade Inversa

Min:

Max:

Quantidade do Valor Anterior: 0

Característica de Tempo:

Relacionamento: Relacionamento06

Tipo:

Relacionamento Inverso:

Cardinalidade

Min:

Max:

Cardinalidade Inversa

Min:

Max:

Quantidade do Valor Anterior: 0

Característica de Tempo:

Relacionamento: Relacionamento07

Tipo:

Relacionamento Inverso:

Cardinalidade

Min:

Max:

Cardinalidade Inversa

Min:

Max:

Quantidade do Valor Anterior: 0

Característica de Tempo:

Relacionamento: Relacionamento08

Tipo:

Relacionamento Inverso:

Cardinalidade

Min:

Max:

Cardinalidade Inversa

Min:

Max:

Quantidade do Valor Anterior: 0

Característica de Tempo:

Relacionamento: Relacionamento09

Tipo:

Relacionamento Inverso:

Cardinalidade

Min:

Max:

Cardinalidade Inversa

Min:

Max:

Quantidade do Valor Anterior: 0

Característica de Tempo:

Relacionamento: Relacionamento10

Tipo:

Relacionamento Inverso:

Cardinalidade

Min:

Max:

Cardinalidade Inversa

Min:

Max:

Quantidade do Valor Anterior: 0

Característica de Tempo:

Relacionamento: Relacionamento11

Tipo:

Relacionamento Inverso:

Cardinalidade

Min:

Max:

Cardinalidade Inversa

Min:

Max:

Quantidade do Valor Anterior: 0

Característica de Tempo:

Relacionamento: Relacionamento12

Tipo:

Relacionamento Inverso:

Cardinalidade

Min:

Max:

Cardinalidade Inversa

Min:

Max:

Quantidade do Valor Anterior: 0

Característica de Tempo:

Relacionamento: Relacionamento01

Tipo:

Relacionamento Inverso:

Cardinalidade

Min:

Max:

Cardinalidade Inversa

Min:

Max:

Quantidade do Valor Anterior: 0

Característica de Tempo:

BNF_Método.txt

Metodo: CadastraVeículo

Descrição:

Classe: Veículo

Tipo do Metodo:

Pre Condição: 0

Pos Condição: 0

Corpo da Ação:

Parâmetros de Entrada:

Parâmetros de Saída:

Metodo: ConsultaVeículo

Descrição:

Classe: Veículo

Tipo do Metodo:

Pre Condição: 0

Pos Condição: 0

Corpo da Ação:

Parâmetros de Entrada:

Parâmetros de Saída:

Metodo: ConsultaProprietário

Descrição:

Classe: Proprietário

Tipo do Metodo: Classe

Pre Condição: 0

Pos Condição: 0

Corpo da Ação:

Parâmetros de Entrada:

Parâmetros de Saída:

Metodo: AlteraCPF

Descrição:

Classe: Veículo

Tipo do Metodo:

Pre Condição: 0

Pos Condição: 0

Corpo da Ação:

Parâmetros de Entrada:

Parâmetros de Saída:

Metodo: AlteraDadosProprietá

Descrição:

Classe: Proprietário

Tipo do Metodo:
Pre Condição: 0
Pos Condição: 0
Corpo da Ação:
Parâmetros de Entrada:
Parâmetros de Saída:

Metodo: ConsultaDadosProprie
Descrição:
Classe: Proprietário
Tipo do Metodo:
Pre Condição: 0
Pos Condição: 0
Corpo da Ação:
Parâmetros de Entrada:
Parâmetros de Saída:

Metodo: ConsultaPorCPF
Descrição:
Classe: Veículo
Tipo do Metodo:
Pre Condição: 0
Pos Condição: 0
Corpo da Ação:
Parâmetros de Entrada:
Parâmetros de Saída:

Metodo: CadastraMulta
Descrição:
Classe: Multas
Tipo do Metodo:
Pre Condição: 0
Pos Condição: 0
Corpo da Ação:

Parâmetros de Entrada:
Parâmetros de Saída:

Metodo: ConsultaMulta
Descrição:
Classe: Multas
Tipo do Metodo:
Pre Condição: 0
Pos Condição: 0
Corpo da Ação:
Parâmetros de Entrada:
Parâmetros de Saída:

Metodo: IndicaTemMulta
Descrição:
Classe: Veículo
Tipo do Metodo:
Pre Condição: 0
Pos Condição: 0
Corpo da Ação:
Parâmetros de Entrada:
Parâmetros de Saída:

Metodo: Licenciamento
Descrição:
Classe: Veículo
Tipo do Metodo:
Pre Condição: 0
Pos Condição: 0
Corpo da Ação:
Parâmetros de Entrada:
Parâmetros de Saída:

BNF_Evento.txt

Evento: Usuário1
Descrição:
Tipo do Evento: Externo
Ativo: Sim
Regra: Regra01

Evento: Usuário2
Descrição:
Tipo do Evento: Externo
Ativo: Não
Regra: Regra02

Evento: ConsultaVeículo
Descrição:
Tipo do Evento: Interno
Ativo: Não
Regra: Regra03

Evento: LicenciamentoAnual
Descrição:
Tipo do Evento: Temporal
Ativo: Não
Regra: Regra05
Tipo de Evento Temporal: Periódico
Método: Licenciamento
Quando: 1
Atraso: 1
Periódico: 1
Relativo:
Tipo: Contínuo

BNF_Trigger.txt

Trigger: T01
Prioridade: 0
Condição: 0
Condição Desabilitada: 0
Sequência de Execução:
Habilitado: Sim
Métodos: CadastraVeículo

Trigger: T02
Prioridade: 0
Condição: 0
Condição Desabilitada: 0
Sequência de Execução:
Habilitado: Não
Métodos: AlteraDadosProprietá

Trigger: T03
Prioridade: 0
Condição: 0
Condição Desabilitada: 0
Sequência de Execução:
Habilitado: Não
Métodos: ConsultaVeículo

Trigger: T04
Prioridade: 0
Condição: 0
Condição Desabilitada: 0
Sequência de Execução:
Habilitado: Não
Métodos: Licenciamento