UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

TESE DE DOUTORADO

Análise por Visões: Uma Técnica para Análise de Requisitos

Autor: José Estevão Picarelli
Orientadora: Profa. Dra. Beatriz Mascia Daltrini

Este exemplar corresponde à redação final da tese defendida por José Estevão Picarelli e aprovada pela Banca Examinadora.
Data: 11/03/2003

Componentes da Banca Examinadora:

Profa. Dra. Beatriz Mascia Daltrini
Prof. Dr. Mario Jino
Prof. Dr. Ivan Luiz Marques Ricarte
Prof. Dr. Eleri Cardozo
Dr. Marcos Carneiro da Silva
Dr. Marcos Lordello Chaim

2003

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

P58a

Picarelli, José Estevão
Análise por visões: uma técnica para análise de requisitos / José Estevão Picarelli. --Campinas, SP: [s.n.], 2003.

Orientador: Beatriz Mascia Daltrini
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Engenharia de software. 2. Sistemas de informação gerencial. 3. Análise de sistemas. 4. Software - Desenvolvimento. I. Daltrini, Beatriz Mascia. II.Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

RESUMO

O objetivo deste trabalho é contribuir para a engenharia de software apresentando três técnicas de análise de requisitos para aplicação no contexto de desenvolvimento de *sistemas de informação.* Uma das técnicas, a *análise por visões,* oferece um modelo referência para definir um particionamento de um *limite* por *visões*. Cada *visão* é caracterizada pelas entidades *foco* e *plano*, que destacam sua abstração. O *foco* caracteriza a natureza dos requisitos com os valores *dado, evento, controle, função e comportamento*. O *plano* caracteriza um domínio com os valores *utilização* da informação, *produção* da informação, *administração* do projeto de software, *software construção* (desenvolvimento), *software validação* (verificação e testes) e *software evolução* (manutenção). A outra técnica, a *análise dos eventos,* oferece um roteiro para determinar e caracterizar os eventos e para modelar a reação de um sistema ou de objetos para cada evento. A técnica *análise de componentes* apresenta um roteiro para que a partir de uma função conceitual detalhada ou das saídas oferecidas por uma aplicação, determinem-se e/ou documentem-se os módulos de software (funcionalidades que serão implementadas com uma linguagem de programação) que serão necessários para a construção de uma nova aplicação de software ou os módulos que uma aplicação de software existente possui.

ABSTRACT

The main goal of this work is to contribute with the software engineering presenting the three techniques of analysis of requirements to be applied in the context of the information systems development. One of the techniques, vision analysis, gives a reference model to establish a division of a vision limit. Each vision is distinguished by the focus and plane entities which demonstrate its abstraction. The focus denotes the nature of requirements with the values: data, event, control, function and behavior. The plane denotes a dominium with the values: information use, information production, and administration of the software project, software construction (development), software validation (checking and tests) and software evolution (maintenance). The other technique, event analysis, presents a direction to define and express the events and to modulate the reaction of a system or objects in each event. The technique component analysis, from a detailed conceptual function or from the exits given by an appliance, presents a direction to define and/or record the software modules (functions that will be carried out through a programming language) that will be necessary for the construction of a new software appliance or the modules that a current software appliance already has.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1 - Introdução

A construção de aplicações de software que suportam *sistemas de informação* é um trabalho complexo que envolve, entre outras coisas, vários tipos de tecnologias, profissionais com qualificações diferentes, equipes relativamente grandes, problemas de natureza interdisciplinar, restrições técnicas e administrativas estreitas e clientes cada vez mais exigentes [Jacobson01].

Devido à alta disponibilidade econômica oferecida pelas indústrias de equipamentos e serviços relativos à tecnologia de informação, a demanda por novas aplicações de software é geometricamente crescente. Esta demanda deve atender a um espectro que vai da computação pessoal até a computação corporativa. Assim, melhorar a capacidade de desenvolvimento, através de melhoria de métodos e processos, é uma preocupação de diversas organizações [Weiss02].

Produtos de software de uso geral tais como: sistemas operacionais, sistemas gerenciadores de banco de dados, editores ou linguagens de programação, possuem, historicamente, um enorme sucesso técnico e comercial. São aplicativos de software de uso genérico e alguns voltados para a operacionalização de determinada tecnologia.

No âmbito de sistemas de informações, há atualmente uma tentativa de se conseguir este mesmo sucesso na construção de aplicações de software de utilidade genérica. São aplicativos de software possíveis de serem compartilhados por diversas pessoas e diversas organizações embutindo e operacionalizando regras de negócio. São as aplicações genericamente referenciadas pela sigla *ERP* (*Enterprise Resource Planning*) [Haberkorn99]. Elas são idealizadas e concebidas levando-se em conta o que as organizações têm em comum e no que elas são iguais. Mas, como a prática tem

demostrado, as organizações bem como as pessoas, possuem mais diferenças do que semelhanças [Weinberg92]. Assim, a implantação dessas aplicações de software de uso genérico têm gerado um trabalho significativo, com alto grau de complexidade de adaptação dessas aplicações não só para uma pessoa mas também para uma organização em particular [Rolland01]. Além disso, mesmo quando esta adaptação tem sucesso, apenas 40% das necessidades de informações são atendidas. A implantação de um sistema de gestão *ERP* custa em média sete vezes mais que a compra das licenças do software. O SAP/R3, por exemplo, tem mais de cinco mil parâmetros, o que evidencia seu grau de complexidade [Scheer00].

Os conhecimentos adquiridos na realização de um projeto de construção ou adaptação de um produto de software não servem aparentemente para o próximo. A tecnologia muda rapidamente e a conclusão do projeto geralmente se arrasta no tempo. Em geral, além das mudanças tecnológicas, mudam também as circunstâncias de desenvolvimento, as equipes de trabalho, a natureza do problema, os perfis dos clientes, os estilos de gestão, os recursos disponíveis e o ambiente de trabalho [Weiss02] entre um projeto de software e outro. Assim, não existindo, na prática, um processo de desenvolvimento de software estabelecido [Avison03], constante e sob controle, as melhorias, os registros de erros e acertos e a transferência de experiências são difíceis de serem realizadas. Além disso, é muito difícil melhorar o improviso [Mann89].

A educação, treinamento e ensino de disciplinas relativas a algumas atividades do processo de desenvolvimento de software requerem esforços especiais em sua condução, principalmente aquelas que, devido a sua própria natureza, agregam pouco formalismo e pouco determinismo. São as atividades de estudos, de elicitação e de análise de requisitos.

Um aspecto que merece destaque é que software não é um produto de uma única natureza. As classificações clássicas dos anos 70 e 80, de software científico ou comercial

[Yourdon89] [Pressman97], software básico ou aplicativo não atendem mais ao amplo leque de diversidade da natureza de software hoje existentes. Embora em [Yourdon88] [Pressman97] e [Jacobson01] as atividades inerentes à construção de produtos de software receberem um tratamento e nomeação única e genérica são, na realidade, atividades que possuem características diferentes, conforme a natureza diferente de cada tipo de software relativo a sua aplicação. Por exemplo, estudar, analisar ou testar uma *folha de pagamento*, um *joguinho de computador*, um *hiperdocumento* ou um *aplicativo de autotreinamento*, exige do profissional capacitação técnica diferente para realizar cada atividade de estudo, análise ou teste. Há diferentes tipos de serviços, diferentes valores éticos, diferentes domínios, diferentes analistas [Wessels01] [Kiedaisch01].

As diversas técnicas de modelagens de requisitos, sedimentadas ou emergentes, estão ainda sendo experimentadas e lapidadas. Neste trabalho é proposta uma forma que possibilite, resgatando a abstração de cada técnica, a aplicação do que cada ferramenta de representação de requisitos tem de bom. Não há a consideração, por exemplo, de diagramas obsoletos, fora de moda ou ruins por si só. Considera-se a aplicação adequada ou não dos mesmos. Do *HIPO* (*Hierarchy Input Processing Output*) [Rocha87] até a *UML* (*Unified Modeling Language*) [Booch99] passando pelo famoso fluxograma [Martin87], todos podem ser bastante úteis.

O interesse por questões metodológicas de análise de requisitos varia com o tempo. As metodologias ganham destaque e maior interesse quando os desenvolvedores voltam-se para o processo de fazer software. Normalmente isto ocorre em épocas de razoável estabilização tecnológica como agora. Hoje, os profissionais de desenvolvimento e implantação de *sistemas de informação* estão sedimentando e disponibilizando aplicações de software, em um aparente consenso tecnológico. Desenvolve-se software para uma arquitetura Cliente/Servidor [Bochenski94][Renaud93], interfaces distribuídas [Barfield93] e customizáveis, gerenciadores de banco de dados relacionais [Date81][Korth86] e disponibilidade e acessos via *Intranets* e *Internet*. O interesse por aspectos metodológicos

pode diminuir quando o mercado disponibiliza novos produtos de software básicos tais como uma nova linguagem de programação, um novo sistema operacional, a proposta de uma nova arquitetura, de um novo gerenciador de banco de dados e outros que fazem sucesso comercial. Daí os desenvolvedores voltam-se para resultados, para produtos finais, e esperam por uma solução tecnológica para seus problemas de desenvolvimento de aplicações de software e não uma solução metodológica.

Os métodos e técnicas de análise de requisitos constituem-se basicamente de técnicas de modelagem [Pressman97]. Além de remover ambigüidades [Coleman94] e possibilitar o registro dos requisitos necessários, esses métodos e técnicas buscam também, através da confecção de modelos interessantes, reduzir a complexidade de uma realidade observada. O analista de sistemas normalmente não cria um sistema mas observa um já existente e, objetivando implementá-lo em uma tecnologia diferente da vigente, elabora modelos que sejam adequados para a tecnologia destino. Como um dos principais objetivos é a redução de complexidade espera-se que os modelos produzidos também sejam simples e claros. Um ponto que pode contribuir para tal simplicidade é que, cada modelo contemple a representação de apenas um tipo de abstração. Este, entre outros, é um ponto pretendido pela *Análise por Visões.*

Os pontos clássicos citados como mitos do software [Pressman97], apontados como sendo as principais causas que dificultam o bom andamento do trabalho na área de software são:

1.O usuário ou cliente não sabe o que quer,
2.Tudo muda muito e o tempo todo,
3.O cliente nunca se satisfaz, quanto mais o atendemos mais ele quer,
4.Tudo é complicado e, como os prazos são curtos, não há tempo para análises,
5.O alto volume de manutenção não libera recursos para execução de melhorias.

Esses pontos não representam problemas mas sim características inerentes ao trabalho de desenvolvimento de software. Um desafio é ajudar as pessoas a compreender o que elas querem de um sistema de informação [Wessels01] e isto é o que justifica a presença de um analista [Argila98]. A Teoria Geral dos Sistemas [Bertalanffy75] mostra que as coisas mudam independente de nossas vontades. Tratar a volatilidade dos requisitos é outro desafio no desenvolvimento de software [Antón01]. Como a informação alimenta-se dentro de um ciclo de retroalimentação positivo (e não negativo), quanto mais informação se tem, mais informação necessita-se. Quanto à questão de complexidade, uma das vocações da análise de sistemas é a redução de complexidade. Através da remoção de ambigüidades [Kovitz02] e redução de complexidade é que as atividades de análise tentam gerar benefícios. Quanto à manutenção excessiva e como software não desgasta [Pressman97], não precisaria ser mantido. A maioria das atividades efetuadas sob o nome de manutenção são atividades de reparos [Cox86] [Weinberg85].

Comentados estes pontos, outros apresentam-se como diagnóstico mais adequado para os fatores que realmente atrapalham e, podem comprometer o bom resultado do trabalho de desenvolvimento de uma aplicação de software:

1. Tentativa de se copiar uma realidade,
2. Sobrecarga de detalhes,
3. Incoerência entre abstração e representação,
4. Projeto tecnológico prematuro.

A *tentativa de se copiar uma realidade* vem do fato da não percepção da diferença entre *reproduzir* uma realidade e *produzi-la* idêntica [Bennaton84]. Dois modelos são considerados equivalentes quando um reproduz o comportamento do outro. É uma questão

de conveniência o quanto a reprodução e o original se confundem. O alto volume da chamada manutenção de software pode ter suas raízes nesta tentativa de cópia. Manter atualizada uma cópia de uma realidade, seja ela qual for, exige um esforço significativo.

A *sobrecarga de detalhes* vem da constatação prática de que até pequenas aplicações de software possuem um volume bastante expressivo de detalhes. Os profissionais podem sentir-se perdidos pelo excesso de informações envolvidas. Os detalhes necessitam ser levantados e organizados de maneira cuidadosa, de forma gradual e em momentos corretos.

A *incoerência entre abstração e representação* pode ser notada através da avaliação da qualidade dos modelos produzidos e da conseqüente qualidade das saídas finais, tais como relatórios, telas ou planilhas, geradas por uma aplicação de software.

O *projeto tecnológico prematuro* nasce normalmente e decorre de ansiedade, pressa e/ou pressão administrativa. Ocorre quando se apresenta uma solução tecnológica para um problema que ainda não foi formulado ou não foi ainda bem compreendido. Muitos projetos de software têm falhado porque os requisitos são pobremente negociados entre os envolvidos [Olson01].

Dentro deste cenário é que se enquadra o trabalho aqui apresentado.

Como objetivo geral este trabalho tem a intenção de poder contribuir na área da *Engenharia de Software* esperando que ela se torne realmente um dia uma *engenharia* no verdadeiro sentido da palavra [Champeaux02].

Como objetivos específicos, dentro do contexto da análise de requisitos, pretende-se apresentar as técnicas de particionamento que seguem:

. **Análise por Visões** - que procura, através da apresentação do conceito de **visão**, oferecer uma alternativa para a escolha adequada de representação dos requisitos que uma aplicação de software deve contemplar. **Visão** representa uma entidade onde, para um **limite** que define um contexto, são agregados dois componentes: o **foco**, que caracteriza uma natureza de requisitos e o **plano**, que caracteriza um domínio.

. **Análise dos Eventos** - que procura, para **foco** com valor **evento**, facilitar, através da proposta de um guia e de uma representação gráfica, a modelagem dos eventos e das reações por eles provocadas.

. **Análise de Componentes** - que procura, através da proposta de um roteiro de modelagem, tratando a tecnologia de informação de forma genérica, apresentar um conjunto limitado de operações que implemente qualquer funcionalidade, dando assim um passo em direção às atividades de projeto.

O trabalho está dividido em 5 capítulos:

No Capítulo 1, *Introdução*, apresentam-se o cenário onde o trabalho está inserido, os problemas de desenvolvimento de software cujas técnicas de análise de requisitos propostas pretendem auxiliar nas soluções.

No Capítulo 2, apresentam-se as principais abstrações das técnicas/métodos de análise de requisitos já sedimentadas e das emergentes. Encerra-se o capítulo com uma apresentação das mais importantes ferramentas de representações gráficas e textuais.

No Capítulo 3, apresentam-se em detalhes a *Análise por Visões*, a *Análise dos Eventos* e a *Análise de Componentes*.

No Capítulo 4, apresentam-se algumas recomendações e sugestões para representações de *Limite*, e também representações para os vários tipos de *Foco,* conforme alguns valores de *Plano* na abordagem sistêmica e na orientada a objetos.

No Capítulo 5, apresentam-se as conclusões e as sugestões para novos trabalhos.

No Anexo I há um estudo de caso onde se aplicam as técnicas apresentadas.

CAPÍTULO 2 - Análise de requisitos

Há atividades, no contexto da engenharia de software, que são inerentes à disciplina de engenharia de requisitos. São as atividades para descobrir, documentar e manter o conjunto de requisitos de um sistema. Por engenharia entende-se que as técnicas para tal podem ser utilizadas de forma sistemática e repetitiva. No contexto de desenvolvimento de sistemas comerciais/industriais, o termo análise de sistemas tem mais ou menos o mesmo sentido [Sommerville97].

A engenharia de requisitos é caracterizada como atividades *up-front* cujo objetivo é determinar os requisitos correta, consistente e completamente. Se isto não for bem realizado é alto o risco de fracasso do projeto [Greenspan01].

As descrições das funções e restrições são os requisitos para o sistema. O processo de descobrir, analisar, documentar e verificar essas funções e restrições cabe à engenharia de requisitos. O termo requisito não é utilizado de forma consistente [Sommerville01]. Há, entre outras definições, como sendo uma necessidade que deve ser declarada de forma bastante abstrata para não induzir em uma solução predefinida [Davis93]. Há também como sendo declarações de serviços ou restrições de um sistema a ser desenvolvido [Kotonya98].

Encontram-se subdivisões das atividades da engenharia de requisitos como elicitação, análise, especificação, validação e gestão de requisitos [Kotonya98][Thayer97]. Encontra-se também, o termo análise de requisitos sendo tratado de forma ampla e abrangente, contemplando atividades de extração, definição e validação das necessidades do cliente, que devem ser supridas pelo sistema de informação [Eva01].

Dá-se aqui preferência para o termo análise de requisitos. Ela representa um fator critico de sucesso no processo de desenvolvimento de aplicações de sistemas de informação. A análise de requisitos envolve a descoberta de problemas a serem resolvidos, regras de negócios a serem utilizadas, e o método de apresentação das informações para

usuários [Seilheimer00]. Se mal executada, a análise de requisitos pode levar a boas soluções técnicas para o problema errado [Boehm01].

Há casos onde só as atividades de análise de requisitos podem durar 15 meses [Dusire02] e a especificação de requisitos chegar a 2.000 páginas [Kiedaisch01].

Segundo pesquisa realizada em 1995 pelo *Standish Group* [Penna00] envolvendo 365 empresas americanas, somente 16,2% dos projetos de aplicação de software foram concluídos dentro do orçamento e prazos previstos, contemplando apenas 42% das características e funcionalidades desejadas. Estima-se que, naquele ano, foram gastos US$ 81 bilhões em projetos cancelados e US$ 59 bilhões além dos custos previstos. Uma pesquisa mais recente realizada na Europa, foi efetuada em 3.800 organizações de 17 países. Ela julgou como maior problema no desenvolvimento de aplicações de software, a baixa qualidade da especificação de requisitos [Lamsweerde00]. Em 2002, como resultado de pesquisa realizada também em empresas americanas, foram constatados que 20% dos projetos de sistemas de informação fracassam e que 46% dos projetos são concluídos com redução das funcionalidades ou com extrapolação de prazos e/ou custos [Procaccino02].

De acordo com Kumar [Kumar02] dos quatro principais riscos de fracasso em um projeto de desenvolvimento de software, dois têm a ver diretamente com a análise de requisitos. São por ele apontados como os principais riscos: 1. as especificações das necessidades incompletas, devido à complexidade ou falta de conhecimento das regras de negócios; 2. o entendimento errado das especificações; 3. os cronogramas e orçamentos intangíveis e 4. a pouca experiência da equipe técnica.

O correto entendimento dos requisitos é uma necessidade chave para o sucesso do desenvolvimento de um sistema de informação. Esta compreensão é problemática devido às limitações cognitivas e diferença de vocabulário [Alvarez02][Almendros02] entre as pessoas envolvidas. Muitos projetos de software têm falhado pois os requisitos são pobremente negociados entre os envolvidos [Olson01].

Apesar dos gerentes de projetos de software terem a sensação de que, ultimamente as coisas melhoraram [Heales00], eles acreditam na necessidade de elaboração de novas técnicas, métodos e ferramentas para tratamento de requisitos, que contribuam para minimizar a gravidade do problema da baixa qualidade de especificação de requisitos [Weiss02]. Vários métodos orientados para a reusabilidade e utilização de padrões têm sido propostos [Gamma95][Larman99][Knethen02].

2.1. Considerações preliminares

Normalmente, técnica e método são utilizados como sinônimos. Necessitamos de uma ampla diversidade de técnicas pois hoje se desenvolve uma variedade muito maior de software [Sommerville01]. Assim, frente à diversidade apenas a diversidade [Ashby70] [Bertalenffy75][Weinberg94].

Segundo Kettinger [Kettinger97] há mais de 72 técnicas usadas só para acompanhar o mapeamento de processos. Nos últimos 20 anos um grande número de métodos têm sido propostos. De 1988 a 1998, foram propostos mais de 19 métodos orientados a objetos só na forma de livros [Wieringa98]. Mesmo assim, eles são, por várias causas, pouco adotados [Riemensch02]. Uma pesquisa, em 1999, no Reino Unido, mostrou que apenas 57% das organizações adotavam algum tipo de metodologia [Avison03].

Uma técnica/método contempla um conjunto de idéias que a caracteriza. Essas idéias dizem respeito a sua abstração. Abstrair é observar um fenômeno e destacar e/ou ignorar aspectos irrelevantes, focando em aspectos essenciais, conforme interesse e conveniência do observador [Brown97]. Não há abstração sem representação nem o inverso [Takahashi90]. A representação é a forma que, através de uma linguagem, ou seja, de convenções sintáticas e semânticas adotadas, explicita-se uma abstração.

Existe uma interdependência entre linguagem e pensamento [Hayakawa78]. Aceita-se que a linguagem condiciona o pensamento e vice-versa, e, nenhuma precede a outra. Linguagem e pensamento moldam-se mutuamente [Worf56].

Esta interdependência entre linguagem e pensamento explica em parte o porquê da existência de defensores fanáticos, como se fossem verdadeiras doutrinas ou religiões [Yourdon86], em torno de uma linguagem de programação, de um método de desenvolvimento de software ou de determinada notação gráfica [Sommerville01].

Geralmente, como forma de representação são utilizados dois tipos de linguagens que se complementam: uma textual e uma gráfica. Algumas explicações para isto são fundadas na vocação diferenciada de cada hemisfério do cérebro humano. Um, relativo ao intelecto, que trata melhor as palavras, enquanto outro, relativo às emoções, que trata melhor as imagens [Guyton89][Goleman96][Franquemont02]. A imagem bate nos olhos; o texto bate direto na mente [Saffo93]. Imagens, gráficos, palavras e textos se complementam [Moore01].

Os dois tipos de linguagens de repesentação são utilizados na análise de requisitos. As técnicas de análise de requisitos são técnicas de modelagem [Booch94][Pressman97] [Booch99][Jacobson01]. Conforme o nível de detalhes, a estratégia de modelagem pode ser do geral para o particular (*top-down*) ou vice-versa (*bottom-up*). A explicação para esses valores de estratégia pode estar na psicologia que afirma que nosso pensamento é um contínuo entremear de processos de análise (decomposição) e síntese (composição) não havendo prevalência de um sobre outro [Witehead71][Weinberg88].

Essas técnicas de modelagem da análise de requisitos buscam em heurísticas, práticas de sucesso, opiniões e experiências pessoais, sua matéria prima para dar-lhes substância. Produzem, de forma pouco formal, modelos cuja qualidade está diretamente ligada às visões, cultura e habilidades dos modeladores [Weinberg93]. Desenvolver software é, em essência, a criação do formal a partir do informal [Kovitz02].

As técnicas/métodos de análise de requisitos foram sendo mais ou menos utilizadas no decorrer dos anos. Os anos 60 e 70 foram conhecidos como a era pré-metodológica. A ênfase era apenas em programação. A tecnologia de informação disponível era considerada viável para informatizar processos administrativos repetitivos, aqueles possíveis de serem representados por algum algoritmo. Não havia o conceito, como se entende hoje, de usuário ou cliente que expressam suas necessidades informacionais. Os desenvolvedores, em um trabalho individualista, raramente entendiam dos negócios organizacionais [Avison03].

No final dos anos 70, principalmente com a tecnologia de banco de dados, viabiliza-se o oferecimento de informações para tomada de decisões. A busca de eficiência nos negócios, levantamento de reais necessidades e documentação dos programas, entre outros, passam a ser grandes preocupações. Caracteriza-se a era nascente das metodologias. Surge o desenvolvimento baseado em fases/etapas, o *SDLC – Systems Development Life Cicle*, sendo o mais famoso o modelo cascata (*waterfal-model*) [Pressman92]. Nessas fases/etapas são então associadas técnicas particulares e, às vezes, tratadas também questões de gerenciamento. Os anos 80 foram a era de proliferação das metodologias [Avison03].

Normalmente, as principais técnicas de análise e projeto emprestam seus adjetivos dos paradigmas de programação. Baseado no paradigma de programação surgem inicialmente as técnica de projeto (*design*) e, na seqüência, as técnica de análise. Foi assim com a análise estruturada, análise orientada a objetos e, está sendo agora com a emergente orientação a aspectos (*aspect-oriented*) [Elrad01].

Essas técnicas/métodos diferenciam-se em alguns pontos e, em outros, podem possuir as mesmas características. Em [Wieringa98] encontram-se várias propostas diferentes para comparação e avaliação das técnicas/métodos. Várias características foram obtidas deste trabalho para sugerir aqui os 8 itens básicos que podem caracterizar a abstração de uma técnica/método. Assim, na seção 2.2 e 2.3 algumas das técnicas/métodos de análise de requisitos são apresentadas segundo sua abstração. Propõe-se aqui que a abstração, que é o que fundamenta o paradigma de cada técnica, seja caracterizada pelos seguintes itens:

1. Natureza predominante: método, técnica.
2. Orientação: dados, processos, objetos, outro.
3. Escopo: sistema, assunto, domínio, outro.
4. Critério de particionamento: função, dado, evento, objeto, outro.
5. Baseado em: ciclo de vida, *frameworks, guidelines,* outro.
6. Estratégia: *top-down*, *bottom-up,* outra.
7. Modelagem: conceitual, tecnologia.

Segue uma descrição de cada item e seus valores.

1. Natureza predominante: método, técnica, outro.

Entende-se por técnica a maneira de fazer algo e por método o modo de aplicar essa técnica. O método oferece uma seqüência de passos a serem dados, um caminho a percorrer. O método pode agregar questões de gerenciamento e controle, distribuição de tarefas e de responsabilidades. Uma técnica pode ter vários métodos e vice-versa. Segundo [Wieringa98] um método de especificação de requisitos deve ter: uma ou mais técnicas, regras para interpretação da técnica, regras de interconexão, e heurísticas para uso da técnica. Há porém métodos sem vínculo a alguma técnica específica de análise de requisitos. Neste caso, o método pode tratar apenas de questões administrativas e do processo, tais como orientações para formação de grupos e equipes e não tratar de questões técnicas de decomposição e/ou organização de requisitos.

2. Orientação: dados, processos, objetos, outro.

Orientação, tanto no sentido de *a partir de*, como no sentido de *em direção a,* é utilizada para adjetivar atividades genéricas, como análise, projeto e programação, por exemplo. Este termo é utilizado, às vezes, para titular a própria técnica/método. Dentro de uma orientação genérica pode-se ter várias técnicas/métodos. Encontram-se também os

valores de orientação a objetivos, orientação a equipes, orientação a usuários e outras. Para técnicas de análise de sistema, os valores de dados, processos e objetos são amplamente utilizados.

3. Escopo: sistema, assunto, domínio, outro.

O escopo de uma técnica/método indica basicamente como a abstração da abrangência para a aplicação da técnica é determinada. O valor sistema indica que uma fronteira bem definida é estabelecida. O valor assunto indica que apenas um tema foi definido. O valor domínio indica um conjunto de assuntos, tendo todos algo em comum que nomeiam este domínio específico. O valor domínio pode ter também algo a ver com um ou mais tipos de problemas específicos que se pretendem resolver. Outros valores podem ser programa, negócio, empresa, departamento, etc.

4. Critério de particionamento: função, dado, evento, objeto, outro.

O critério de particionamento indica a forma de análise, de decomposição propriamente dita, de divisão adotada sobre um escopo. Outros valores podem ser objetivo, módulo, etc.

5. Baseado em: ciclo de vida, *frameworks, guidelines,* outro.

Este item indica como a técnica/método insere-se no processo de desenvolvimento de software como um todo. A técnica/método pode pertencer a alguma fase/etapa particular de um modelo de ciclo de vida. A técnica/método pode ser um modelo referência do tipo de *frameworks* que representam estruturas genéricas onde, para cada projeto, instanciam-se as particularidades. A técnica/método pode ser apresentada por *guidelines* que representam boas recomendações, esquemas, valores e roteiros utilizados como referência e como guia para o projeto.

6. Estratégia: *top-down*, *bottom-up,* outra.

A estratégia de detalhamento indica o caminho de como os níveis de detalhes são observados e modelados. A estratégia não indica os vários níveis de abstração em diferentes domínios e sim, os níveis de detalhes. A estratégia estabelece o caminho para se efetuar o refinamento e/ou condensação de modelos. O valor *top-down* para a estratégia indica que se parte do geral em direção ao particular e, o valor *bottom-up* indica que se executa o caminho inverso.

7. Modelagem: conceitual, tecnologia, ambas.

A modelagem de valor conceitual indica a pretensão de se retratar o problema e não a solução de tecnologia de informação adotada ou a adotar; significa uma modelagem isenta de tecnologia. A modelagem de valor tecnologia indica que se pretende representar os aspectos tecnológicos no modelo.

2.2. Técnicas/métodos sedimentados

Ao longo dos anos muitas técnicas e métodos têm sido propostos para realizar a análise de requisitos. A seguir, estão descritas as técnicas/métodos que mais se destacaram, no contexto de desenvolvimento de sistemas de informações, no sentido de serem amplamente usadas e de terem maior repercussão entre os analistas. Ao fim de cada descrição das técnicas/métodos sua abstração é analisada conforme os itens apresentados na seção anterior. Existem outras técnicas/métodos que são mais conhecidas em contextos específicos como, por exemplo, o de desenvolvimento de software básico ou o de desenvolvimento de aplicações de sistemas de tempo real. A maioria são denominadas com os nomes de seus autores, tais como Ward/Mellor [Ward85], Hatley/Pirbhai [Hatley87], Martin [Martin88], Yourdon [Yourdon89], Jackson [Masiero92] e são basicamente extensões das técnicas que possuem como critério de particionamento dominante [Ossher01] as funções, dados, controles, eventos ou objetos. A maioria dos analistas ou engenheiros de software considera essas versões como *dialetos* das técnicas/métodos mais conhecidos [Pressman92].

Segue a apresentação de técnicas que foram elaboradas para a análise de requisitos propriamente dita, tais como: a Análise Estruturada; o SADT; a Análise Essencial; a Análise de Dados; o Use-Case; a Análise Orientada a Objetos. De outras técnicas que foram adaptadas para a análise de requisitos, tal como o HIPO. De outras ainda que complementam a análise de requisitos com informações administrativas, roteiros e orientações de trabalho, tais como: o FAST; o IDEF; o JAD; a Prototipação e a Análise de Domínios.

HIPO - Hierarchy plus Input-Process-Output

Desenvolvido pela IBM, o *HIPO* foi proposto originalmente como uma ferramenta de documentação de programas [Rocha87]. Representa as entradas, o processo e as saídas através de um documento subdividido em 3 colunas. Dá destaque para a coluna do processo, a coluna do meio, pois esta é bem mais larga que as colunas das entradas e das

saídas. Isto reflete a alta preocupação predominante na época, anos 70, com os algoritmos, com os processos, em comparação aos dados, as entradas e saídas. Este documento é associado como uma ferramenta de programação ou, no máximo, como uma ferramenta para projeto. Porém, a hierarquia, representando uma subdivisão funcional e, tendo esta a forma de um organograma, gráfico associado mais a questões organizacionais do que de programação, pode-se caracterizar o *HIPO* como uma ferramenta de análise. O *HIPO* consta apenas dessas duas representações. Sua abstração consiste de uma decomposição funcional hierárquica e de uma especificação da função de forma detalhada.

A abstração do *HIPO* pode ser caracterizada como segue:

1. Natureza predominante: técnica.
2. Orientação: processos.
3. Escopo: outro (programa).
4. Critério de particionamento: função.
5. Baseado em: outro (programação).
6. Estratégia: *bottom-up*.
7. Modelagem: tecnologia

FAST - Facilitated Application Specification Techniques

O *FAST* [Csilag85] visa a aproximar os usuários e clientes com a equipe de trabalho dos desenvolvedores. Esta equipe tem como objetivo identificar os problemas, negociar possíveis soluções e especificar um conjunto básico de requisitos [Pressman97]. Através de encontros periódicos e planejados busca-se o consenso e compreensão uniforme dos problemas e requisitos. Primeiramente, através de lista de perguntas e respostas tenta-se definir o escopo e justificar, de forma priorizada, a aquisição ou desenvolvimento de um novo produto. Após isto, são levantados na seguinte ordem: listas de objetos (do ambiente, produzidos pelo sistema, e utilizados pelo sistema), serviços (processos ou funções), restrições e critérios de desempenho. Essas informações são representadas em listas, descrições livres e esquemas gráficos elaborados livremente.

Enquadrando o *FAST* como sendo mais aplicável em atividades da engenharia de requisitos que precedem à análise dos mesmos, pode-se caracterizar sua abstração como segue:

1. Natureza predominante: método.
2. Orientação: processos.
3. Escopo: assunto, domínio.
4. Critério de particionamento: outro (objetivo).
5. Baseado em: *guidelines.*
6. Estratégia: *top-down.*.
7. Modelagem: conceitual.

SADT - Structured Analysis and Desing Technique

O *SADT* [Ross84] contempla o particionamento de um sistema em funções e dados sendo aplicável nas etapas de análise e projeto. Utiliza como representação gráfica os diagramas de atividades (*actigram*) e de dados (*datagram*). Para cada atividade (retângulo) representam-se os dados de entrada (lado esquerdo), dados de saídas (lado direito), os dados de controle (lado superior) e o processador da atividade (lado inferior). Não se representa a assincronia entre atividades (no DFD isto é efetuado através de depósitos). Em outro modelo, para cada dado(s) (retângulo) representa-se a atividade geradora (lado esquerdo), a atividade que o utiliza (lado direito), a atividade de controle (lado superior) e o dispositivo de armazenamento do dado(s) (lado inferior).

Pode-se caracterizar a abstração do *SADT* como segue:

1. Natureza predominante: método, técnica.
2. Orientação: dados, processos.
3. Escopo: sistema.
4. Critério de particionamento: função, dado.

5. Baseado em: ciclo de vida.
6. Estratégia: *top-down.*.
7. Modelagem: conceitual.

IDEF - Integrated computer aided manufacturing DEFinition

O *IDEF* é derivado do *SADT* e foi desenvolvido pela necessidade do governo americano de rever, reestruturar e organizar várias agências federais [Mayer95]. É constituído de vários modelos, sendo o primeiro o *IDEF-0,* o modelo funcional. São os outros modelos: 1-Modelo de informações; 1X-Modelo de dados; 2-Modelo de simulação; 3-Descrição de processos; 4-Modelos de objetos; 5-Modelo de coleta/aquisição de informação; 6-*Design;* 7-Sistema de Informação; 8-Modelo de interface do usuário; 9-Modelo de cenários; 10–Modelo de arquitetura de implementação; 11-Modelo de construção; 12-Modelo organizacional; 13-Projeto de mapeamento do esquema triplo; 14-Projeto de rede.

Pode-se caracterizar a abstração do *IDEF* como segue:

1. Natureza predominante: método, técnica.
2. Orientação: dados, processos.
3. Escopo: sistema, assunto, domínio.
4. Critério de particionamento: função, dado, objeto, outro (objetivo).
5. Baseado em: ciclo de vida.
6. Estratégia: *top-down.*.
7. Modelagem: conceitual, tecnologia.

AE - Análise Estruturada

A análise estruturada tornou-se conhecida principalmente através de Tom DeMarco [DeMarco78], Page-Jones [Jones80] e Chris Gane [Gane82]. Na aplicação da análise estruturada determina-se inicialmente a abrangência de um sistema definindo, o que é o sistema, e qual a parcela do ambiente que com ele interage. A partir daí, através de sucessivos refinamentos, modelam-se as funções que tratam os dados do sistema. Existe a preocupação de integrar as funções do sistema. Exige-se que seja representado apenas "o que" fazer (problema) e não "o como" fazer (solução tecnológica). Como representação gráfica utiliza-se o *DFD - Data Flow Diagram* e como complementação de informações o Dicionário de Dados e a Especificação de Função. Apesar de utilizar o mesmo adjetivo, *estruturada*, empregado na programação, sua abstração é totalmente diferente. As idéias da programação estruturada (três estruturas de controle: seqüência, decisão e repetição) [Dijkstra68] são aplicadas para a elaboração das especificações das funções.

Pode-se caracterizar a abstração da análise estruturada como segue:

1. Natureza predominante: método, técnica.
2. Orientação: processos.
3. Escopo: sistema.
4. Critério de particionamento: função.
5. Baseado em: ciclo-de-vida.
6. Estratégia: *top-down.*.
7. Modelagem: conceitual.

AES - Análise Essencial

A análise essencial [McMenamim84] [Yourdon89] pode ser considerada uma extensão da análise estruturada. Mudam-se alguns termos básicos, como lógica (*o que*) para essência, e física (*como*) para encarnação. Propõe-se que o primeiro particionamento seja efetuado, observando o ambiente do sistema, através da determinação dos eventos.

Considerando um mecanismo de evento/respostas, determinam-se as reações funcionais do sistema, que podem ser: a geração de resposta ao ambiente, o armazenamento de dados ou a mudança de estado de alguns objetos do sistema. As atividades do sistema que não serão informatizadas (*ad hoc*) são representadas no ambiente do sistema e a parte planejada, que será informatizada, é categorizada em funções fundamentais e funções custodiais. Custodiais são aquelas funções necessárias para dar apoio às fundamentais estabelecendo e mantendo a memória do sistema.

Pode-se caracterizar a abstração da análise essencial como segue:

1. Natureza predominante: método, técnica.
2. Orientação: processos.
3. Escopo: sistema.
4. Critério de particionamento: função, evento.
5. Baseado em: ciclo de vida.
6. Estratégia: outra (nível intermediário).
7. Modelagem: conceitual.

JAD - Joint Application Design

O *JAD* [August91] [Chin95] é um método desenvolvido em 1977 e depurado em 1984 pela IBM, que visa a formar e orientar equipes no sentido de acelerar o processo de desenvolvimento de sistemas de software. Define-se, buscando o consenso, desde os objetivos e requisitos até as telas e relatórios, principalmente sob ponto de vista dos clientes e usuários. O *JAD* é baseado em 4 princípios: dinâmica de grupo, recursos visuais, processo organizado e documentação *WYSIWYG (what you see is what you get*). Dentro de cada tarefa JAD/PLANO e JAD/PROJETO, aplicam-se as fases de customização, reuniões e fechamento.

Pode-se caracterizar a abstração do *JAD* como segue:

1. Natureza predominante: método.
2. Orientação: processos.
3. Escopo: sistema, domínio.
4. Critério de particionamento: evento, outro (objetivo).
5. Baseado em: *guidelines*..
6. Estratégia: outra (tarefa).
7. Modelagem: conceitual.

ADA - Análise de Dados

A análise de dados cuja modelagem é baseada no *ERA (Entity-Relation-Attribute)* é a técnica mais amplamente utilizada [Sommerville01]. Proposto inicialmente por [Chen77] o modelo de entidades e de relacionamentos apresenta as entidades de dados, seus atributos associados e as relações entre as entidades. O relacionamento entre entidades pode também ter atributos associados. Modelam-se os dados de forma independente das funções que os manipularão. As estruturas e elementos de dados que caracterizam os atributos das entidades/relacionamentos podem também ser especificados em um Dicionário de Dados.

Pode-se caracterizar a abstração da análise de dados como segue:

1. Natureza predominante: técnica.
2. Orientação: dados.
3. Escopo: domínio.
4. Critério de particionamento: dado.
5. Baseado em: ciclo-de-vida.
6. Estratégia: *bottom-up* (modelo geral só no fim)
7. Modelagem: conceitual, tecnologia.

PP - Prototipação

O paradigma da prototipação (*Prototyping Paradigm)* é recomendado quando há significativa dificuldade em expressar os requisitos de um sistema. Um protótipo é uma versão operacional inicial de um software para apoiar as atividades de levantamento e validação de requisitos [Sommerville01]. O protótipo é construído em várias rodadas do ciclo de 3 atividades: ouvir o cliente, construir/revisar a maqueta (*mock-up*) e testar/validar a maqueta [Pressman97]. O protótipo é uma versão rápida e ruim (*quick and dirty*) que é progressivamente aperfeiçoada até o requisito ter a aceitação do cliente [Melendez90].

Pode-se caracterizar a abstração da prototipação como segue:

1. Natureza predominante: método.
2. Orientação: processos (funcionalidade).
3. Escopo: sistema.
4. Critério de particionamento: função.
5. Baseado em: ciclo de vida.
6. Estratégia: *bottom-up* (nível de aplicação).
7. Modelagem: tecnologia (através dela mostrar o resto)

UC - Use Case

O *UC* é utilizado para redução de um contexto através de um particionamento funcional [Jacobson92]. Tem larga aceitação entre os desenvolvedores principalmente devido a sua simplicidade . A decomposição é efetuada apenas por funcionalidades visíveis a um ator qualquer. Ator é uma entidade que assume papéis lógicos. Em um diagrama, as funcionalidades são representadas por elipses e os atores por uma silhueta que lembra um boneco. A relação entre o ator e as funcionalidades são indicadas através de segmentos de retas.

Pode-se caracterizar a abstração da análise funcional por *use-cases* como:

1. Natureza predominante: técnica.
2. Orientação: processo.
3. Escopo: domínio.
4. Critério de particionamento: função.
5. Baseado em: ciclo de vida.
6. Estratégia: *bottom-up* (por ator).
7. Modelagem: conceitual, tecnologia.

AOO - Análise Orientada a Objetos

A AOO agrupa objetos (metáfora de dados mais processos encapsulados) com características comuns, em classes de objetos (definições de estruturas e operações). Funcionalidades são obtidas através de seqüências de mensagens entre objetos (metáfora para comunicação). Um domínio é particionado em classes de objetos. Utilizam-se basicamente dois tipos de modelagem: a estática (características) e a dinâmica (comportamento) para representar os objetos/classes [Rumbaugh91]. Muitos métodos adaptam elementos das técnicas estruturadas para a abstração de objetos tais como extensões do modelo entidades e relacionamentos [Mellor88] [Coad91], coesão e acoplamento [Jones95], desenvolvimento incremental e outros [Martin92]. Pode-se representar objetos/classes agindo em sociedade e/ou individualmente. É amplamente utilizado como linguagem padrão de representação dos modelos a *UML – Unified Modeling Language* [Booch99].

Pode-se caracterizar a abstração da AOO como segue:

1. Natureza predominante: método, técnica.
2. Orientação: objetos.
3. Escopo: assunto, domínio.
4. Critério de particionamento: objeto.
5. Baseado em: ciclo de vida.

6. Estratégia: *bottom-up*.
7. Modelagem: conceitual, tecnologia.

ADO - Análise de Domínios

A análise de domínios [Arango89] tem a preocupação básica de propor maneiras para identificar, classificar, catalogar e divulgar objetos visando a sua potencial reusabilidade [Firesmith93]. O analista de domínio teria o papel principal de um *administrador de objetos*, tal qual tivemos, nos anos 80, o *administrador de dados* [Pressman97]. Mellor [Mellor92], por exemplo, sugere três categorias para os domínios: aplicação, arquitetura e implementação. Jones [Jones95] [Jones00] sugere a classificação, numa ordem de baixo para alto grau de reutilização, de aplicação, negócio, arquitetura e base. Agrupar requisitos em um mesmo domínio pode também facilitar as atividades de desenvolvimento e manutenção de aplicações [John02].

A abstração da análise de domínios pode ser caracterizada como segue:

1. Natureza predominante: método.
2. Orientação: objetos.
3. Escopo: domínio.
4. Critério de particionamento: objeto.
5. Baseado em: ciclo de vida.
6. Estratégia: *bottom-up*.
7. Modelagem: conceitual, tecnologia.

2.3. Técnicas/Métodos emergentes

Além da busca de melhorias na aplicação e de padrões de uso das técnicas/métodos já sedimentadas, novas técnicas/métodos estão surgindo no contexto de desenvolvimento de sistemas de informação. Atualmente, o que se pretende encontrar são técnicas/métodos que possibilitem principalmente a rapidez no desenvolvimento das aplicações [Avison03].

Segue a apresentação das técnicas *GOR* e *VBRE*, aplicáveis em alto nível de abstração; da *XP*, que pode ser caraterizada mais como uma estratégia para equipes desenvolverem aplicações; a *AOP*, que ainda se encontra no nível de programação.

GOR - Goal-Oriented Requirements

GOR é uma técnica utilizada para elicitação, análise e documentação dos requisitos de um sistema [Lamsweerde02]. Propõe o ciclo: aquisição de objetivo (em diversos níveis de abstração); refinamento de objetivo; derivação de requisitos e a checagem semi-formal de consistência/completeza [Lamsweerde01]. A especificação de requisitos está completa se todos os objetivos foram contemplados por esta especificação [Letier00]. A avaliação da capacidade de desenvolvimento realizada em uma organização pode ser efetuada sob o critério de quantidades de objetivos contemplados [Weiss02]. No contexto da *GOR* os objetivos são mais estáveis e mais perenes que os requisitos pois esses implementam os objetivos [Letier00]. O objetivo pode ser representado via um *template* para declaração de seus atributos tais como nome, referência, especificação, prioridade, utilidade e flexibilidade [Lamsweerde01]. Podem-se utilizar esquemas gráficos ligando objetivos com objetivos e objetivos com requisitos.

A abstração da *GOR* pode ser caracterizada como segue:

1. Natureza predominante: método.
2. Orientação: processos.
3. Escopo: domínio.

4. Critério de particionamento: outro (objetivo).
5. Baseado em: ciclo-de-vida.
6. Estratégia: *top-down.*
7. Modelagem: conceitual.

VBRE - Viewpoint-based Requirements Engineering

VBRE é uma técnica que objetiva resolver problemas decorrente da multiplicidade de pessoas e interesses, certamente dependentes e conflitantes, que tomam parte no processo de engenharia de requisitos [Silva02]. Há muitos pontos de vista (*multiple stakeholders viewpoint*) [Lamsweerde00] que devem ser considerados. A análise baseada em pontos de vista reconhece a existência de várias perspectivas e oferece um *framework* para descobrir conflitos nos requisitos propostos [Sommerville01]. A análise baseada em pontos de vista reconhece que é improvável a descoberta de todos os requisitos de um sistema considerando apenas uma única perspectiva (ponto de vista) [Sommerville97][Lee02]. Ela oferece maneiras de tratar as inconsistências e informações incompletas obtidas de múltiplas fontes [Easterbrook01]. Normalmente, para representar as diversas fontes de pontos de vista utilizam-se diagramas hierárquicos de especialização. Para representar as informações sobre o ponto de vista e o serviço (funcionalidade) utilizam-se planilhas padrão (*templates*).

A abstração da análise baseada em pontos de vista pode ser caracterizada:

1. Natureza predominante: método, técnica.
2. Orientação: processos.
3. Escopo: sistema.
4. Critério de particionamento: função.
5. Baseado em: *frameworks, templates.*
6. Estratégia: *top-down.*
7. Modelagem: conceitual.

XP - eXtreme Programming

XP [Beck99] tem como princípios: o desenvolvimento incremental; entrega rápida de pequenas funcionalidades; o envolvimento do cliente; a melhoria do código; e a programação impessoal [Sommerville01]. Programação impessoal é a prática onde dois programadores (*pair programming*), trabalhando lado a lado, colaboram no mesmo projeto, código e teste [Williams00][Williams01]. Nesta forma de trabalho, constataram maior prazer no mesmo, melhor produtividade e melhor qualidade no código [Muller01] [Nagappan03]. A *XP* é recomendada para pequenos grupos (até 20 membros) em pequenos projetos onde as tarefas podem ser decompostas em *histórias do cliente* (*story card*) [Lippert01]. A história do cliente é tratada aqui como objetivos, eventos e funções.

A abstração da *XP* pode ser caracterizada como segue:

1. Natureza predominante: método.
2. Orientação: processos.
3. Escopo: assunto, domínio.
4. Critério de particionamento: função, evento, outro (objetivo).
5. Baseado em: *frameworks.*
6. Estratégia: *bottom-up* (aplicação).
7. Modelagem: conceitual, tecnologia.

AOP - Aspect-Oriented Programming

AOP é um paradigma de programação pós objeto [Elrad01] que tem como preocupação básica a implementação e manutenção de requisitos/restrições que atravessam e permeiam (*crosscutt*) a aplicação como um todo. Com esta característica esses tipos de

requisitos são denominados aspectos. Aspectos podem surgir quando, em tempo de projeto, feito primeiramente uma decomposição dominante, eles não puderam ser implementados em apenas uma função, classe ou módulo. Para sua implementação foram quebradas fronteiras de uma estrutura [Coad01] fazendo com que esta característica fique espalhada (*scattered)* na modularização de implementação adotada. Cada aspecto pode ser indicado por meio de pontos de sinalizações (*pointcut*) no código implementado para permitir juntar essas características (*join point)* [Kiczales01] [Lieberherr01] e organizá-las em um programa (*weaver*) que destaca e controla determinado aspecto. A *AOP* objetiva um melhor controle nas mudanças, melhorias, adaptações e evolução de software no tratamento de requisitos do tipo aspectos [Trilnik02]. Apesar da idéia de um requisito essencial ter que ser fragmentado para acomodar aptidões e capacidades de um processador (*crosscutting concern*) já ter sido levantada na Análise Essencial [McMenamim84], pesquisas para se usar este paradigma da *AOP* na engenharia de requisitos ainda são imaturas [Rashid02].

Pode-se caracterizar a abstração da *AOP* como segue:

1. Natureza predominante: técnica.
2. Orientação: objetos.
3. Escopo: sistema.
4. Critério de particionamento: outro (aspecto).
5. Baseado em: *frameworks.*
6. Estratégia: *top-down.*.
7. Modelagem: conceitual, tecnologia.

A Tabela 2.1 é o resumo da caracterização da abstração efetuada para cada técnica/método apresentada em 2.2 e em 2.3.

Tabela 2.1 - Resumo da abstração das várias técnicas/métodos

Sigla	1	2	3	4	5	6	7
tec/met	**natureza**	**orientação**	**escopo**	**partic.**	**baseado**	**estratégia**	**modelagem**
HIPO	técnica	processos	programa	função	programa-ção	bottom-up	tecnologia
FAST	método	processos	assunto domínio	objetivo	guidelines	top-down	conceitual
SADT	método técnica	dados processos	sistema	função dado	ciclo de vida	top-down	conceitual
IDEF	método técnica	dados processos	sistema assunto domínio	função dado objeto objetivo	ciclo de vida	top-down	conceitual tecnologia
AE	método técnica	processos	sistema	função	ciclo de vida	top-down	conceitual
AES	método técnica	processos	sistema	função evento	ciclo de vida	pontual	conceitual
JAD	método	processos	sistema domínio	evento objetivo	guidelines	tarefa	conceitual
ADA	técnica	dados	domínio	dado	ciclo de vida	bottom-up	conceitual tecnologia
PP	método	processos	sistema	função	ciclo-vida	bottom-up	tecnologia
UC	técnica	processos	domínio	função	ciclo-vida	bottom-up	conceitual tecnologia

Tabela 2.1 - Resumo da abstração das várias técnicas/métodos (continuação)

Sigla	1	2	3	4	5	6	7
tec/met	**natureza**	**orientação**	**escopo**	**partic.**	**baseado**	**estratégia**	**modelagem**
AOO	método técnica	objetos	assunto domínio	objeto	ciclo-vida	bottom-up	conceitual tecnologia
ADO	método	objetos	domínio	objeto	ciclo-vida	bottom-up	conceitual tecnologia
GOR	método	processos	domínio	objetivo	ciclo-vida	top-down	conceitual
VBRE	método técnica	processos	sistema	função	framework templates	top-down	conceitual
XP	método	processos	assunto domínio	função evento objetivo	framework	bottom-up	conceitual tecnologia
AOP	técnica	objetos	sistema	aspecto	framework	top-down	conceitual tecnologia

As representações, gráficas e textuais, mais conhecidas e utilizadas na aplicação dessas técnicas/métodos são apresentadas na seção 2.4 que segue.

2.4. Representações Importantes

Nesta seção são apresentadas algumas ferramentas de representação gráfica, textual e mista mais conhecidas. São também apresentadas, para as ferramentas de representação textual/mista, a categorização sugerida e o enquadramento dessas ferramentas nesta categorização.

2.4.1. Representações Gráficas

A Tabela 2.2 mostra as ferramentas de representação gráfica apresentadas.

Tabela 2.2 - Referência e nome de representações gráficas

Referência	Nome
DFD	Diagrama de Fluxo de Dados
DFC	Diagrama de Fluxo de Controle
DCU	Diagrama de Caso de Uso
DCL	Diagrama das Classes
DAC	Diagrama de Associação de Classes
DAT	Diagrama de Atividades
DES	Diagrama Estruturado
DSQ	Diagrama de Seqüência
DER	Diagrama de Entidades e Relacionamentos
DTE	Diagrama de Transição de Estados
DDI	Diagrama de Diálogo
DCO	Diagrama de Colaboração
DPA	Diagrama de Pacotes
DNS	Diagrama de Nassi- Shneiderman
DJA	Diagrama de Jackson
DBL	Diagrama de Blocos
DLS	Diagrama de Limite do Sistema

A Figura 2.1 mostra um exemplo de uma parte de um DFD de uma função.

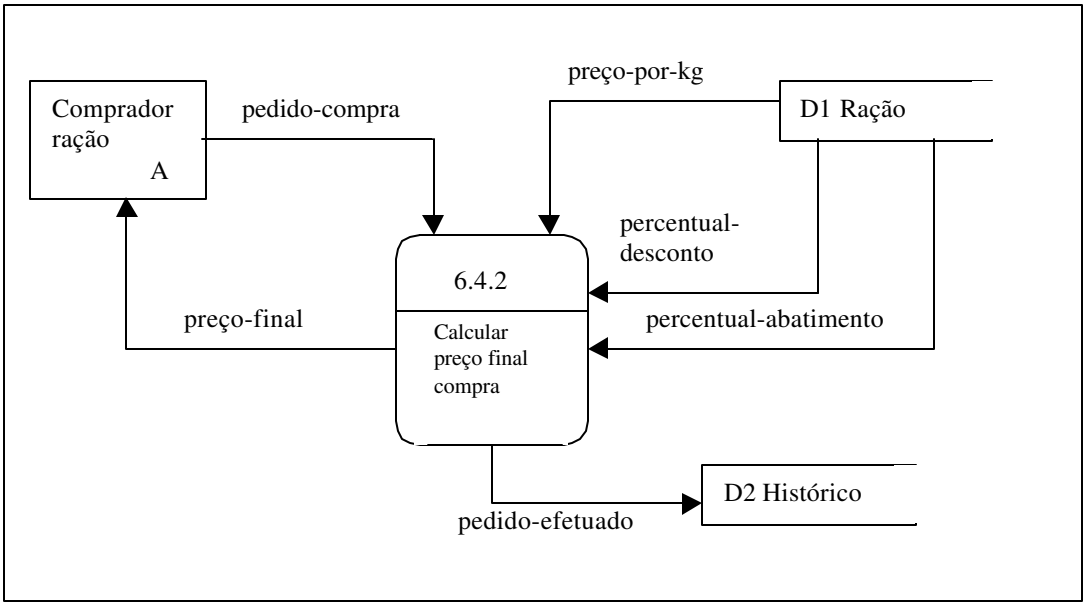


Figura 2.1 - Exemplo de Diagrama de Fluxo de Dados

O DFD - Diagrama de Fluxo de Dados é um diagrama bastante conhecido entre os modeladores de requisitos de sistemas de informação. Foi apresentado no final da década de 70 através da chamada *Análise Estruturada* [Gane79][Demarco78]**.** É recomendado para construção de modelos conceituais, chamados de modelos lógicos, ou seja, modelos isentos de tecnologia. A *Análise Estruturada* está na categoria das metodologias orientadas a processo. O DFD é útil para representar um particionamento funcional. O detalhamento funcional pode ser efetuado, numa estratégia *top-down*, em níveis de DFDs.

O DFC - Diagrama de Fluxo de Controle é o próprio Diagrama de Fluxo de Dados com um elemento a mais, no caso, o Fluxo de controle. Este fluxo pode ser representado através de segmentos de reta pontilhados e com um sentido orientado para a função que receberá o controle.

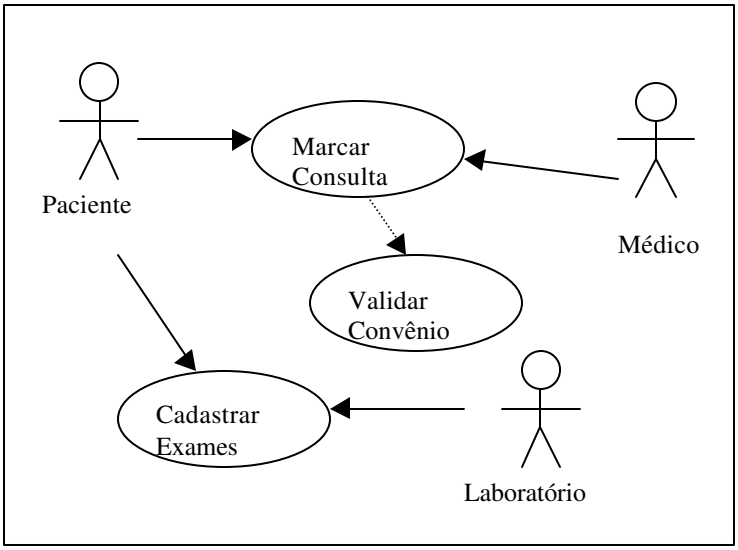


Figura 2.2 - Exemplo de Diagrama de Caso de Uso

O DCU - Diagrama de Caso de Uso é um diagrama proposto inicialmente por Jacobson [Jacobson92] e posteriormente incorporado à UML [Booch99]. Este diagrama tem, relativamente a outros diagramas, uma forte aceitação nos meios profissionais da TI, a Tecnologia da Informação [Walton89]. Uma das razões para isto pode ser sua simplicidade**.** Possui basicamente apenas três elementos e quase nenhuma regra ou formalismo para sua elaboração. Seus elementos são: os *Atores*, os *Casos de Uso* e as *setas*. Este diagrama é recomendado para se realizar um primeiro particionamento simples e geral de um contexto qualquer. Para cada *Ator*, tenta-se esgotar as funcionalidades visíveis a ele. A Figura 2.2 mostra um exemplo de um Diagrama de Caso de Uso.

Para se obter uma simplificação gráfica e uma melhor estética, o Diagrama de Classes [Rumbaugh91] pode ser representado em dois diagramas: O DCL - Diagrama das Classes e o DAC - Diagrama de Associação de Classes.

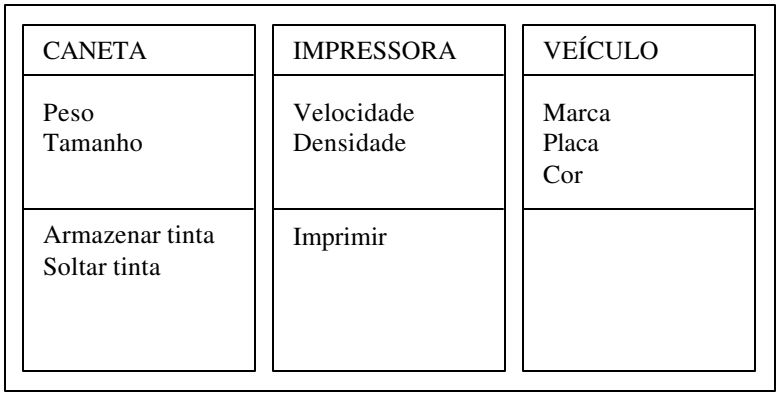


Figura 2.3 - Exemplo de Diagrama das Classes

Para cada classe de uma possível lista elabora-se uma entrada no DCL - Diagrama das Classes. Este diagrama possui apenas um elemento gráfico, um retângulo dividido em três setores, representando a classe. No primeiro setor coloca-se apenas o nome da classe. No segundo setor colocam-se as características estáticas da classe. No terceiro setor colocam-se as funções executadas por esta classe, ou seja, as operações desta classe. Utilizam-se classes desde para se capturar o vocabulário que faz parte do domínio do problema até para se representar itens de software que fazem uma implementação.

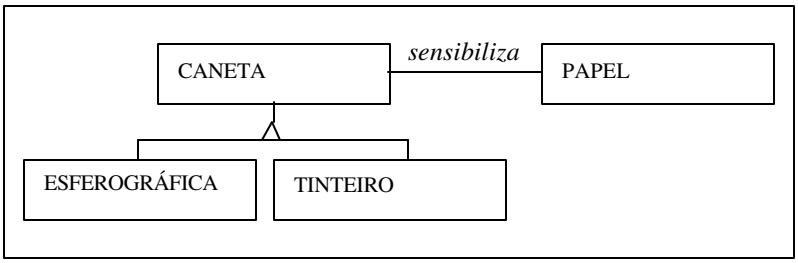


Figura 2.4 - Exemplo de Diagrama de Associação de Classe

O DAC - Diagrama de Associação de Classes representa os possíveis relacionamentos semânticos entre as diversas classes. Ele possui dois elementos: as *Classes* e as *ligações entre as Classes*. As ligações são de três tipos: associação, agregação e especialização/generalização. A(s) associação(ções) representa(m), através de um verbo, um relacionamento semântico entre classes. A especialização/generalização é um relacionamento *é um* ou *é uma*. Isto representa a herança e que é uma questão fundamental e central no contexto de objetos [Harel02]. A adoção de critérios não consistentes, no uso do mecanismo de especialização, no contexto de aplicacões de software, pode levar a futuras dificuldades de manutenção [Schrefl02]. A agregação é um relacionamento de composição, relacionamento *todo-parte.* A Figura 2.4 mostra um exemplo deste diagrama

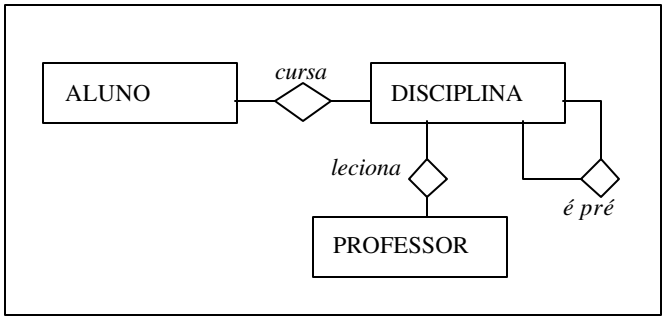


Figura 2.5 - Exemplo de Diagrama de Entidades e Relacionamentos

O DER – Diagrama de Entidades e Relacionamentos [Chen77] é um diagrama onde as entidades são representadas por um retângulo nomeado por substantivos. Os relacionamentos são representados por losangos e também nomeados. Ambos, entidades e relacionamentos, podem ter atributos. A Figura 2.5 mostra um exemplo de um DER.

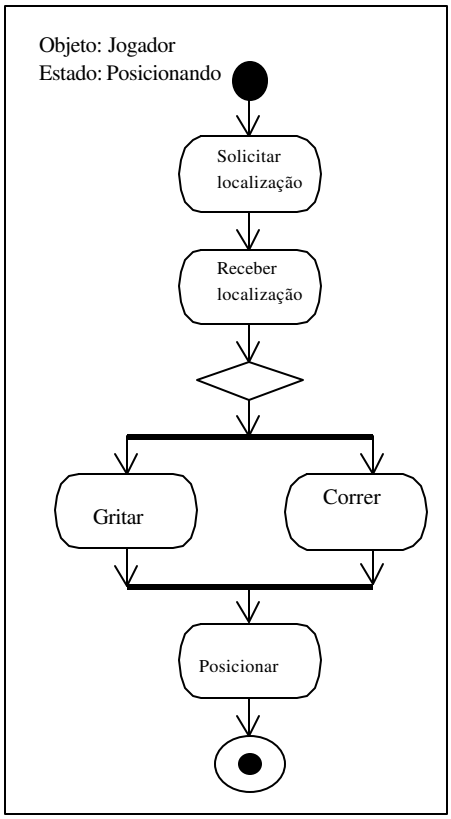


Figura 2.6 - Exemplo de Diagrama de Atividades

O DAT - Diagrama de Atividades é um diagrama, incorporado também à UML [Booch99], que visa a representar o fluxo de uma atividade a outra. Este diagrama faz parte da modelagem do comportamento básico do sistema. A Figura 2.6 mostra um exemplo de um Diagrama de Atividades.

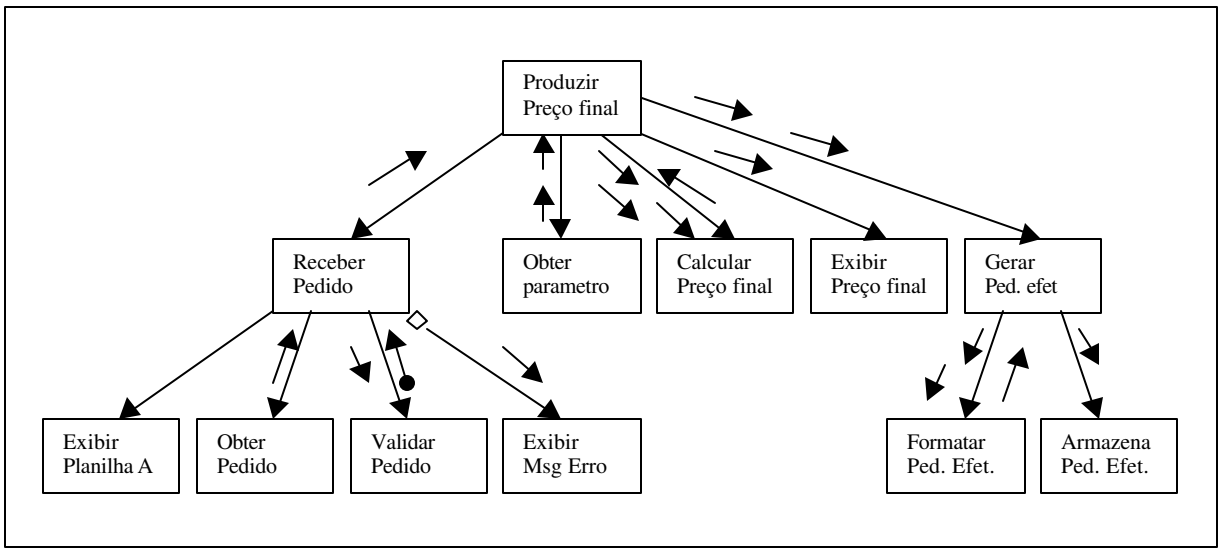


Figura 2.7 - Exemplo de Diagrama Estruturado

O DES - Diagrama Estruturado representa as ligações hierárquicas e comunicações de dados ou controle entre módulos de software [Jones80][Stevens81]. Cada retângulo significa um módulo. A comunicação de dados é representada por setinhas, orientada para o destino. A Figura 2.7 mostra um exemplo de um Diagrama Estruturado.

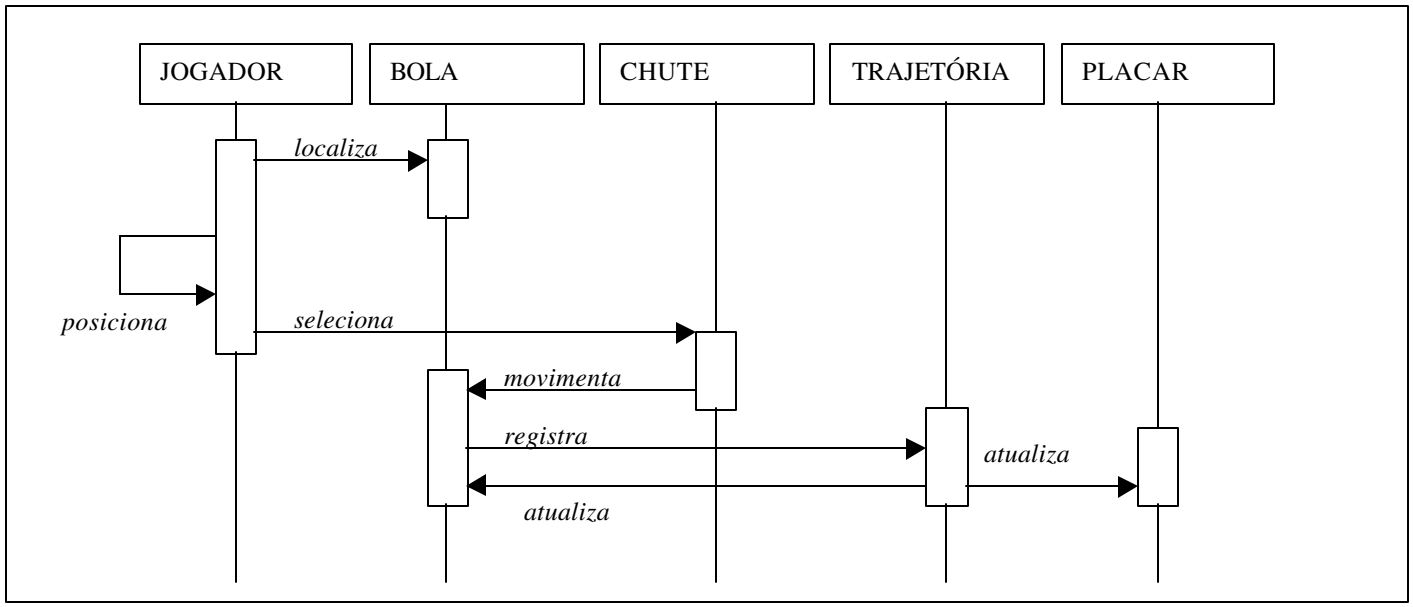


Figura 2.8 - Exemplo de Diagrama de Seqüência

O DSQ - Diagrama de Seqüência é um diagrama já incorporado à UML [Booch99] que representa a seqüência cronológica de eventos e/ou mensagens entre um grupo de objetos para se atingir a funcionalidade especificada em um cenário. Os objetos são colocados no eixo "X" e as mensagens, em ordem crescente no tempo, no eixo "Y". Este diagrama é útil para verificar se todas operações, estimuladas pelas mensagens estão presentes e que são necessárias para que se alcance a funcionalidade desejada. A Figura 2.8 mostra um exemplo do Diagrama de Seqüência.

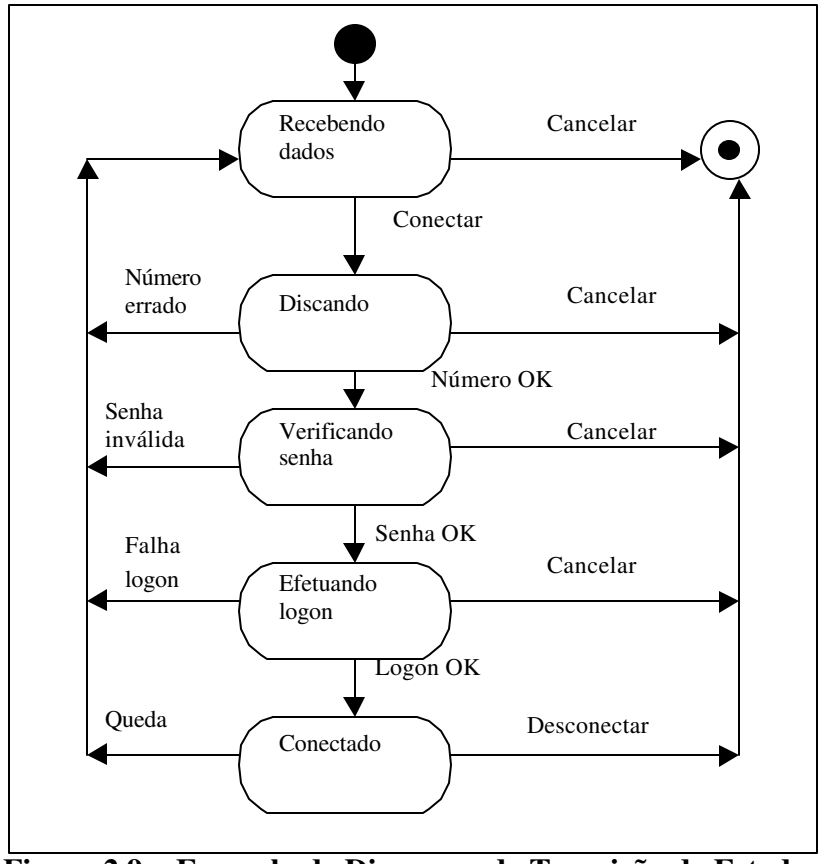


Figura 2.9 – Exemplo de Diagrama de Transição de Estados

O DTE - Diagrama de Transição de Estados é uma máquina de estados finitos. Representam-se os estados através de retângulos, nomeados internamente através de verbo e substantivo ou um adjetivo. A transição é representada através de setas, direcionada no sentido do novo estado, nomeadas pelos eventos ou ações que provocam a mudança. Os estados podem ser subdetalhados em outros estados, permitindo assim que este diagrama seja utilizado tanto na abordagem sistêmica, quando tratado os macroestados, ou na abordagem de objetos, representando os estados de um determinado objeto.

A Figura 2.9 mostra um exemplo de comportamento de um módulo de software, que retrata a funcionalidade de *efetuar conexão à rede* de computadores, via rede telefônica.

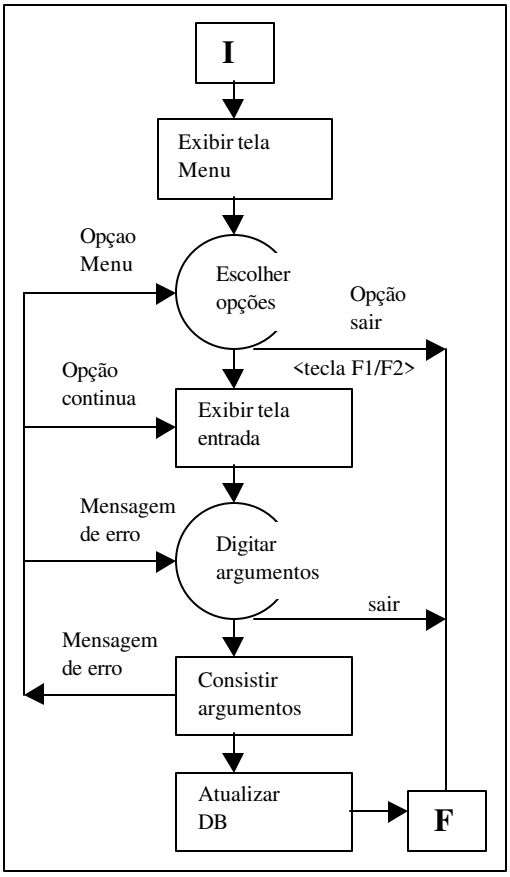


Figura 2.10 - Exemplo do Diagrama de Diálogo

O diálogo entre usuário e aplicação pode ser representado através de um DDI - Diagrama de Diálogo. Na parte interna do retângulo coloca-se o nome da operação e na parte interna da circunferência a ação esperada do usuário. Para indicar o caminho escolhido pode-se indicar sobre a seta, entre os símbolos matemáticos de maior e menor, a tecla, a opção ou um valor de argumento que direcionou o fluxo naquele caminho. Notar que não faz sentido uma seta ligando duas circunferências. A aplicação não pode mudar de estado sem executar nenhuma atividade. Cada resposta do usuário deve ser tratada pela aplicação. A Figura 2.10 mostra um exemplo deste diagrama.

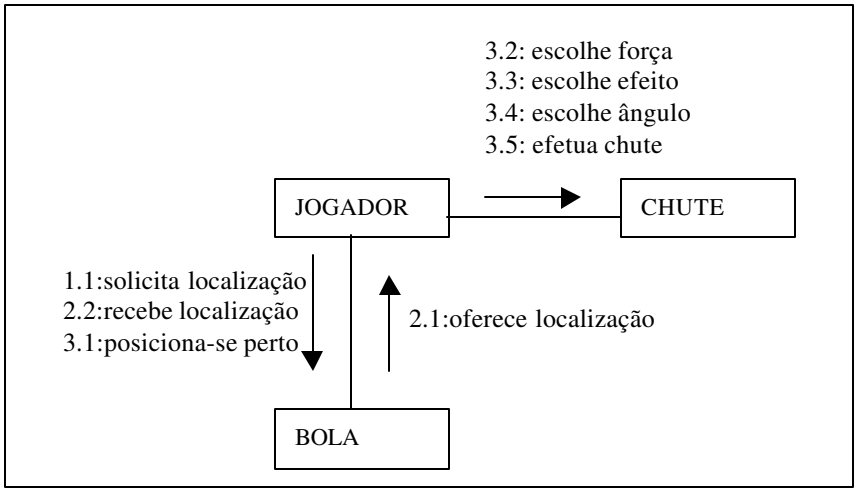


Figura 2.11 - Exemplo de Diagrama de Colaboração

O DCO - Diagrama de Colaboração é um diagrama já incorporado à UML [Booch99] que mostra uma sociedade de objetos que participam de uma interação. Apresenta visualmente a passagem de controle entre eles, numerados seqüencialmente em ordem crescente e explicitada, textualmente ao lado, a seqüência de operações ou atividades executadas pelo objeto para realizar esta passagem de controle. A Figura 2.11 mostra um exemplo de um diagrama deste tipo.

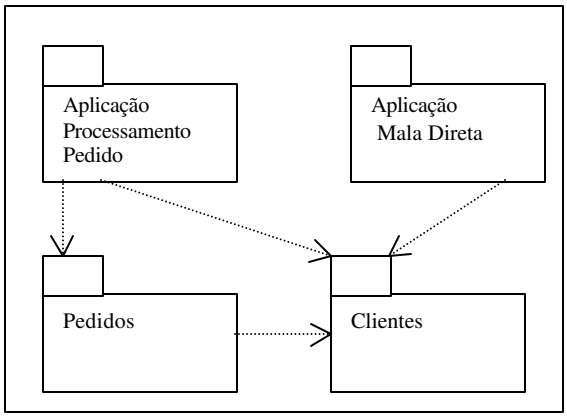


Figura 2.12 – Exemplo de Diagrama de Pacotes

O DPA - Diagrama de Pacotes é um diagrama também já incorporado à UML [Booch99] e é utilizado para auxiliar o agrupamento de classes e suas dependências em

macrofuncionalidades. Os retângulos com uma aba retangular significam os pacotes (*package*) e os fluxos suas dependências. A Figura 2.12 mostra um exemplo de um diagrama deste tipo.

O DNS - Diagrama de Nassi-Shneiderman [Nassi73] é também conhecido como cartas NS. É um diagrama onde não é permitida a representação de desvios. São utilizados para projeto de programas. A seqüência, interação e condição são representadas dentro de uma área retangular como mostra a Figura 2.13.

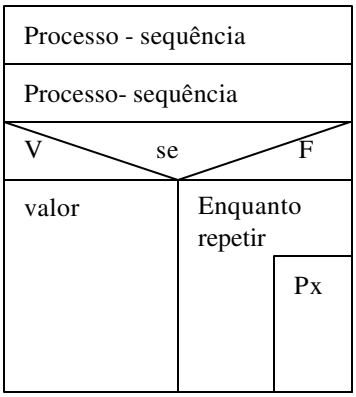


Figura 2.13 - Exemplo de Diagrama de Nassi-Shneiderman

O DJA – Diagrama de Jackson [Jackson81] é um diagrama utilizado normalmente para projetos de programas. Seu método propõe fazer diagramas parciais para tratamento de entradas e de saídas agrupando-os depois, em um único diagrama. Os retângulos indicam os processos. O asterisco representa a repetição e um *o* representa alternabilidade. A Figura 2.14 mostra um exemplo genérico do Diagrama de Jackson.

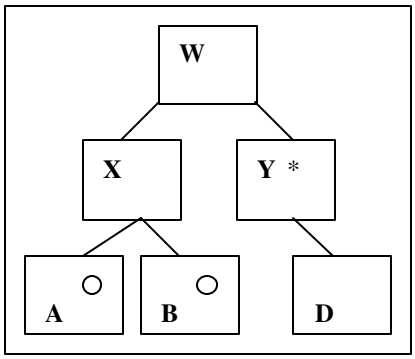


Figura 2.14 - Exemplo de Diagrama de Jackson

O mais conhecido dos diagramas é o DBL - Diagrama de Blocos [Davis83]. As ações ou atividades são representadas por retângulos e a decisão por um losango. A seqüência, que também interliga todos elementos é representada por segmentos de retas com um sentido. A Figura 2.15 mostra um exemplo do DBL - Diagrama de Blocos.

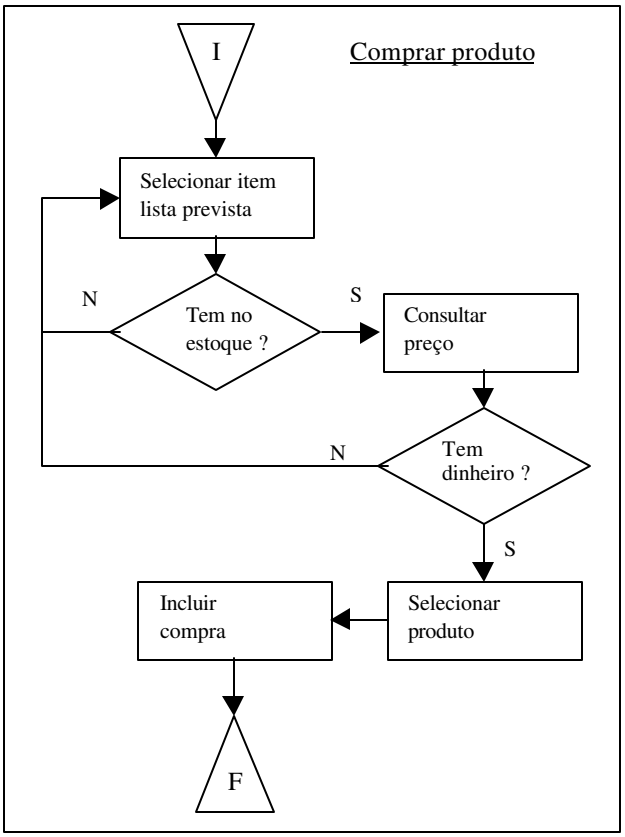


Figura 2.15 - Exemplo do Diagrama de Bloco

O DLS - Diagrama de Limite do Sistema é o diagrama de contexto. Foi sugerido por Gane [Gane79] por meio da tabela *DE / PARA / O QUE*. É a tabela representada graficamente. O *DE* e o *PARA* podem ser o sistema ou alguma entidade externa. O *O QUE* é um fluxo de dados. A Figura 2.16 mostra um exemplo do DLS - Diagrama de Limite do Sistema.

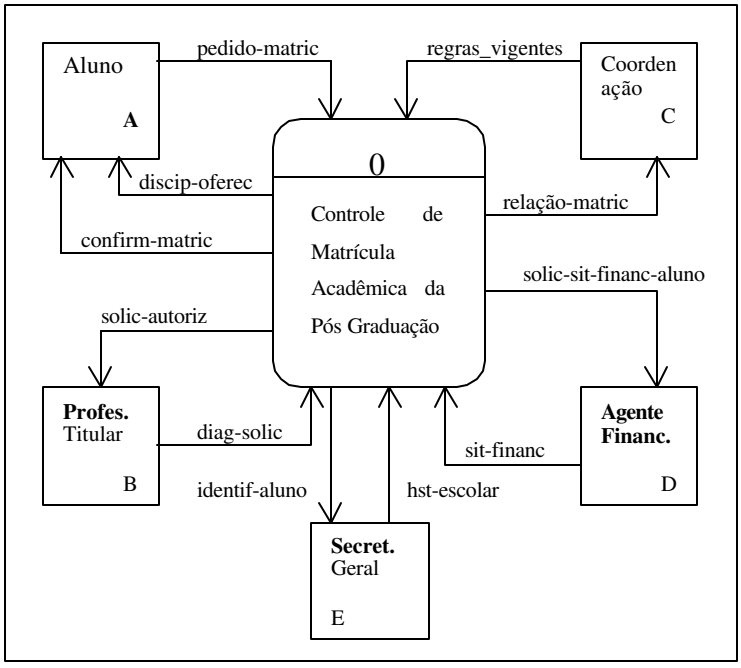


Figura 2.16 - Exemplo de Diagrama de Limite do Sistema

2.4.2. Representação Textual ou Mista

Entende-se por representação textual a utilização da linguagem natural para descrever um modelo. Entende-se por representação mista, a utilização de algum croqui, tabela, esquema ou grafo tendo ou não junto um texto. Para as ferramentas de representação textuais/mista apresenta-se a seguinte classificação: Relação, Descrição, Atribuição e Especificação.

A primeira categoria, *Relação*, consiste simplesmente em construir uma lista de itens. Como exemplo de uma ***Relação***, pode-se ter, como segue, uma *Relação de Eventos* no contexto de uma livraria qualquer:

Relação de Eventos - Livraria

1. Cliente solicita disponibilidade de livro
2. Cliente solicita compra de livro
3. Cliente efetua pagamento de livro comprado
4. Cliente retira livro
5. Fornecedor entrega novo produto
6. Momento de fazer promoções de livros

A segunda categoria, *Descrição*, consiste em definir através da forma narrativa, em texto livre, por meio da linguagem natural. Esta forma de representação só perde, em termos de geração de ambigüidades, para a representação oral. Este fato não importa muito pois essas representações, também chamadas de *Modelos Descritivos*, são normalmente utilizadas apenas como ponto de partida no processo de modelagem e não como uma documentação final.

Como exemplo de uma ***Descrição***, pode-se ter, como segue, uma *Descrição de Função* (modelo descritivo) no contexto de um armazém de vendas de rações:

Descrição Função 6.4.2 - Calcular preço final compra

A ração é voltada para bovinos, eqüinos e suínos.

O cálculo do preço da ração é feito conforme os três componentes básicos, ou seja, o milho, a soja e o cálcio. O cálculo é simples, ou melhor, para a ração-1, composta de 50% de milho e o restante dividido entre soja e cálcio, o preço é de US$ 20 o Kg, sofrendo redução de 15% em compra maior ou igual a 100 Kg. Para a ração-2, contendo 50% de soja e o restante dividido entre cálcio e milho, o preço é de US$ 10 o Kg, com uma redução de 5% em compra acima de 100 Kg (inclusive). A ração-3, composta de 50% de cálcio e o restante entre milho e soja, o preço é de US$ 15 o Kg e 10% de redução em compra maior ou igual a 100 Kg.

Em caso de uma compra de dois ou três tipos de rações que totalize peso maior que 100 Kg, o preço sofre uma redução de 2% do total comprado.

O cliente pode optar ainda por obter um brinde de 100 Kg da ração-1, 150 Kg da ração-2 ou 200 Kg da ração-3, ao comprar por seis meses consecutivos mais de 100 Kg por mês do mesmo tipo de ração.

A terceira categoria, *Atribuição*, considera o objeto a definir como um elemento de uma entidade. A representação é efetuada atribuindo-se valores para os atributos da entidade.

Como exemplo de uma ***Atribuição***, pode-se ter, como segue, o de um de *Elemento de Dado* específico, por exemplo *Sigla-da-UF*. A atribuição pode ser feita como segue:

Atribuição – Sigla-da-UF

1. Nome: *Sigla-da-UF*

2. [Alias]*: *Sigla-do-Estado*

3. [Descrição]: *Duas letras que identificam a Unidade da Federação brasileira*

4.Classe de valor: *Discreto*

5.[Valor]*: *AC, AL, AM, AP, BA, CE, DF, ES, FN, GO, MA, MG, MT, MS, PA, PB, PE, PI, PR, RJ, RN, RO, RR, RS, SE, SC, SP, TO.*

6. Validade: *Dois caracteres alfabéticos e constantes na relação de valores.*

7. [Observação]* *Atribuído pelo Congresso Nacional quando da criação do Estado*.

A quarta categoria, ***Especificação***, representa um passo em direção ao formalismo. Consiste de uma definição efetuada a partir de um *Modelo Descritivo* de uma função. Objetiva auxiliar atividades de teste, validação com clientes e de definição de programas. A

idéia básica é a mesma proposta pela programação estruturada, em que todo algoritmo pode ser especificado utilizando-se apenas as três estruturas de controle, seqüência, decisão e repetição, não existindo a necessidade de instruções de desvios [Dijkstra68]. Uma representação utilizada, no contexto de análise de requisitos, na categoria de *DNL - Disciplined Natural Language* [Smith02], é o chamado *Português Estruturado* [DeMarco78][Gane79].

O *Português Misto* consiste em uma proposta de adaptação do *Português Estruturado* em que, para instruções de repetição, utiliza-se a estrutura **Repetir**-*procedimento*-**para**-*itens*-**para-obter**-*dado* e, para instruções seqüenciais, uma em cada linha, utiliza-se o próprio *Português Estruturado*. Para o conjunto de decisões que leva às ações, as instruções de decisão, utiliza-se a *Tabela de Decisão*. Para uma decisão que leva a ações mutuamente exclusivas, instrução de decisão do tipo caso, utiliza-se a *Tabela Linha-Coluna*.

Tem-se como exemplo a Figura 2.17, que é a ***Especificação*** da função *Calcular preço final de ração*, utilizando-se *Português Misto.*

A Tabela 2.3 mostra o enquadramento de vários itens na proposta de categoria de representação: *Relação, Descrição, Atribuição* ou *Especificação*.

Tabela 2.3 - Representações textuais/mistas utilizadas

Relação	**Descrição**	**Atribuição**	**Especificação**
Eventos	Entidade Externa	Elemento de Dados	Função
Classes	Ator	Estrutura de Dados	Operação
Cenários	Interagente	Fluxo de Dados	Método
	Casos de Uso	Depósito de Dados	Serviço
	Função	Entidade	Atividade
	Funcionalidade	Atributo	
	Cenário	Adjetivo Indefinido	
	Associação	Relacionamento	
	Relacionamento	Associação	
	Ação	Projeto	
	Atividade	Visão	

Figura 2.17 - Especificação Função - Cálculo preço final compra

CÁLCULO **PREÇO_FINAL_COMPRA**

REPETIR	CÁLCULO_PREÇO_RAÇÃO P/CADA TIPO_RAÇÃO	P/OBTER	PREÇO_TP_RAÇÃO
SOMAR	TODOS PREÇO_TP_RAÇÃO	P/OBTER	TOTAL_BÁSICO
SOMAR	TODAS QUANTIDADE	P/OBTER	PESO_TOTAL
EXECUTAR	CÁLCULO_DESCONTO_TOTAL	P/OBTER	DESCONTO_TOTAL
SUBTRAIR	DESCONTO_TOTAL DE TOTAL_BÁSICO	P/OBTER	**PREÇO_FINAL_COMPRA**

CÁLCULO_PREÇO_RAÇÃO

EXECUTAR	CÁLCULO_ QTDE_EXCEDENTE	P/OBTER	QTDE_EXCEDENTE
SUBTRAIR	QTDE_EXCEDENTE DE QUANTIDADE	P/OBTER	QTDE_NORMAL
EXECUTAR	OBTER_PREÇO_KG	P/OBTER	PREÇO-KG
MULTIPLICAR	QTDE_NORMAL POR PREÇO-KG	P/OBTER	PREÇO_NORMAL
EXECUTAR	OBTER-PERC-ABAT	P/OBTER	PERCENT-ABATIMENTO
MULTIPLICAR	QTDE_EXCEDENTE POR PREÇO-KG	P/OBTER	RAÇÃO_EXCED
EXECUTAR	OBTER_PERC_ABAT	P/OBTER	PERCENT-ABATIMENTO
APLICAR	PERCENT-ABATIMENTO EM RAÇÃO_EXCED	P/OBTER	PREÇO_EXCED
SOMAR	PREÇO_NORMAL E PREÇO_EXCED	P/OBTER	PREÇO_TP_RAÇÃO

CÁLCULO_DESCONTO_TOTAL

EXECUTAR	DETERMINAR_PERC_DESCONTO	P/OBTER	PERC_DESCONTO
APLICAR	PERCENTUAL_DESCONTO EM TOTAL_BÁSICO	P/OBTER	DESCONTO_TOTAL

CÁLCULO_QTDE_EXCEDENTE

QUANTIDADE	QTDE_EXCEDENTE
LE 100	ZERO
GT 100	SUBTRAIR 100 DE QUANTIDADE

OBTER_PREÇO_KG

TIPO RAÇÃO	PREÇO-KG
1	$20,00
2	$15,00
3	$10,00

OBTER_PERC_ABAT

TIPO RAÇÃO	1	1	2	2	3	3
QTDE PEDIDA	LE100	GT100	LE100	GT100	LE100	GT100
00%	X		X		X	
15%		X				
10%						X
05%				X		

DETERMINAR_PERC_DESCONTO

PEDIDO_RAÇÃO	EQ 1	EQ 1	GT 1	GT 1
PESO_TOTAL	LE 100	GT 100	LE 100	GT 100
ZERO %	X	X	X	
percentual-desconto				X

2.5. Considerações finais

As diversas ferramentas de representações apresentadas podem ser escolhidas e utilizadas conforme julgamento, conhecimento, habilidade e conveniência do modelador. As diversas técnicas/métodos, já sedimentadas ou emergentes, podem ser combinadas, conforme necessidades e circunstâncias de desenvolvimento. De algum modo, diversas questões administrativas, propostas em vários métodos, também podem ser agregadas de forma combinada para o desenvolvimento de um determinado projeto de software.

A combinação de idéias e boas práticas das diversas técnicas/métodos, bem como inclusões ou adaptações de outras idéias, podem ser efetuadas através de uma ***arquitetura referência*** que não fique baseada em um modelo específico de ciclo de vida o qual determina uma seqüência de fases, um paradigma ou uma técnica particular. A *arquitetura referência* é um modelo padrão, genérico e abstrato, que pode ser usado para qualquer projeto de desenvolvimento de software. Pretende-se então, por princípio, utilizar as mesmas abstrações e representações, já consideradas importantes em um contexto prático de desenvolvimento de sistemas de informação, das técnicas apresentadas.

Algumas técnicas/métodos como a *análise estruturada*, *análise essencial,* o *IDE*F e a *análise por domínios* propõem particionamento de domínios em vários ***planos*** de abstrações como por exemplo a segmentação clássica de lógico e físico. Algumas idéias propostas em métodos tais como o *FAST* (participação de clientes e usuários); o *JAD* (maneiras de montar e conduzir equipes); a *prototipação* (validar através do formato tecnológico); as *orientadas a objetivos* - *GOR* (alinhar requisitos com objetivos); e a *programação extrema – XP* (formação de grupos de alta produtividade na entrega de aplicações) ou em outros *métodos ágeis* de desenvolvimento [Newkirk02][Hirsch02] podem ser também agregadas em um ***plano*** de abstração que trata de questões no contexto da ***administração*** do projeto de software.

É interessante não ficar obrigatoriamente vinculado a um paradigma específico. As diversas ***abordagens*** podem ser utilizadas conforme adequação, natureza do problema e conveniência do modelador. Conforme apresentado, as várias técnicas oferecem as ***abordagens*** básicas da orientação a objetos, um valor do item ***orientação*** e do item ***particionament****o* na Tabela 2.1, e a abordagem sistêmica, um valor do item *escopo* na mesma tabela.

Como mostra a *engenharia de requisitos baseada em pontos de vistas* (*VBRE)*, as várias perspectivas, mesmo gerando modelos semanticamente diferentes, devem ser consideradas. As várias perspectivas que representam os diversos ***focos*** podem ser incorporados em uma mesma técnica/método pois eles contemplam informações que se complementam. Conforme apresentado, os diversos ***focos*** aparecem através do item ***particionamento*** da Tabela 2.1 e podem ter os valores básicos de função, dado e evento.

Conforme mostra a Tabela 2.1 através do item ***escopo***, para as técnicas/métodos apresentadas é importante a determinação de um ***limite.*** Ter um ***limite*** é importante devido a questões, das mais simples como fatores estéticos na confecção de um diagrama, como a questões abstratas de definição de um contexto, assunto ou domínio.

Finalmente, é importante realçar que para a boa qualidade de um modelo de requisitos é fundamental a coerência entre a sua abstração e a sua representação. Uma determinada ***visão*** bem limitada, bem fragmentada e bem caracterizada pode ser útil para destacar-se qual é a abstração que está sendo tratada.

Segue no Capítulo 3 a apresentação detalhada de uma técnica, a *Análise por Visões*, que incorpora os conceitos de ***arquitetura referência, limite, visão, abordagem, foco*** **e** ***plano.***

CAPÍTULO 3 - Análise por Visões

Vários modelos são produzidos nas primeiras etapas da *engenharia de software* [Carvalho01], em atividades chamadas de Estudo, Levantamento, Elicitação e Análise de Requisitos. Além de remoção de ambigüidades [Gause89], uma das principais finalidades de se modelar uma realidade ou parcela dela, sob uma ótica analítica, é a busca de redução de complexidade da mesma. Espera-se que o modelo gerado seja bem menos complexo do que a realidade em si, que o modelo tenha menos variedade [Epstein73] e que, no mínimo, a compreensão da mesma seja facilitada pela observação do mesmo.

Por ótica analítica pretende-se exprimir a idéia de que será efetuada uma divisão do problema em partes adotando algum critério de particionamento. O primeiro critério aqui proposto é de um particionamento por *Visões*. Este particionamento é então denominado de *Análise por Visões*.

O termo *Visão* aqui utilizado difere do conceito de *ponto de vista* apresentado em [Lamsweerde00] e [Silva02]. O conceito de *ponto de vista* consiste basicamente de interpretações diferentes, dadas por diversos tipos de participantes, tais como usuários ou analistas, na observação de um sistema. Difere também do conceito das várias perspectivas, também denominadas visões, necessárias para descrever a arquitetura de um sistema apresentada pela UML [Booch99], tais como visão de caso de uso, visão de projeto, visão de implementação, visão de processo e visão de implantação. *Visão* aqui representa uma entidade que possibilita a caracterização e classificação das necessidades, requisitos e restrições, a que um produto de software deverá atender.

A realidade a ser modelada pode ser limitada e observada sob várias *Visões.* Estabelecido um *Limite*, dado um *Foco* e um *Plano* de observação tem-se uma *Visão*.

O *Limite* é caracterizado e determinado por um assunto, um problema, um tema, uma área de conceitos, um domínio, um conjunto de elementos ou uma curiosidade qualquer que desperte algum interesse de investigação a algum observador.

A escolha de um ou mais *Focos* é efetuada pela necessidade de se destacar por vez, e, de forma separada, um ou mais aspectos relevantes com características semelhantes e que contribuem para alvos comuns.

Os *Planos* são determinados para separar temas ou assuntos sem dependências, oriundos de fontes separadas de requisitos ou de fontes de restrições técnicas e administrativas e que se podem complementar. A sugestão da separação de tipos de requisitos ou restrições em *Planos* vem do fato de que um dos fatores de complexidade aparente de um determinado problema é a mistura de dois ou mais assuntos diferentes. Em qualquer contexto, basta misturarem-se dois ou mais assuntos diferentes que ele se torna aparentemente complicado, sem o ser na realidade.

Cada *Visão* torna-se então o alvo a ser observado, a ser investigado, a ser trabalhado, a ser, enfim, modelado. Essa visão pode ser representada, em um ou mais tipos de modelos, cada qual com maior ou menor grau de detalhes conforme conveniência do modelador. A preocupação pelo levantamento e registro dos detalhes pode ir de um nível macro e geral a um nível micro e específico, *top-down*, ou, ao contrário, de particularidades para as generalidades, *bottom-up* [Weinberg75].

Para definir uma *visão*, ou seja, a partir de um *limite*, escolher um *plano* e centralizar um *foco*, pode-se optar por abstrair dentro dos dois paradigmas: a *abordagem* sistêmica ou pela orientação a objetos. Em cada passo da definição e especificação das características de uma *visão*, a abstração poderá ser efetuada baseada nos conceitos de sistemas ou de objetos. A escolha e aplicação dessas alternativas de abstração depende da natureza e complexidade do fenômeno, do problema ou do contexto a ser observado e, algumas vezes, também das circunstâncias em que essa observação será efetuada.

Definir exatamente como é o começo de qualquer tarefa não é simples. Parece que o princípio e a finalização de todos empreendimentos humanos são desorganizados [Yourdon89]. Exatamente o início de qualquer tarefa, além de não ocorrer de uma única

maneira, é constituído de atividades nebulosas, aparentemente caóticas, difíceis de serem caracterizadas o que leva a pensar que elas simplesmente começam começando.

Uma metodologia de desenvolvimento de software apresenta qual é o passo inicial, indica por onde começar. Após isto, apresenta outro e mais outro em uma seqüência pré estabelecida. Há várias sugestões. A clássica é o modelo cascata (*waterfall*) [Pressman92][Avison03]. Ele apresenta várias etapas em seqüência tais como estudo, análise, projeto, programação, teste e manutenção. Outro modelo é apresentado na chamada Engenharia de Requisitos [Sommervile97] que efetua uma divisão por atividades, tais como elicitação, análise, especificação, validação e gerenciamento [Thayer97] [Kotonya98].

A *Análise por Visões* pode ser utilizada em qualquer metodologia. Parte do princípio de que a decisão de um produto de software a ser desenvolvido já foi tomada. Considera também que recursos de hardware, software e humanos necessários para tal desenvolvimento já foram alocados.

A Figura 3.1 mostra um diagrama de classes na notação UML [Booch99], representando as entidades envolvidas na *Análise por Visões*.

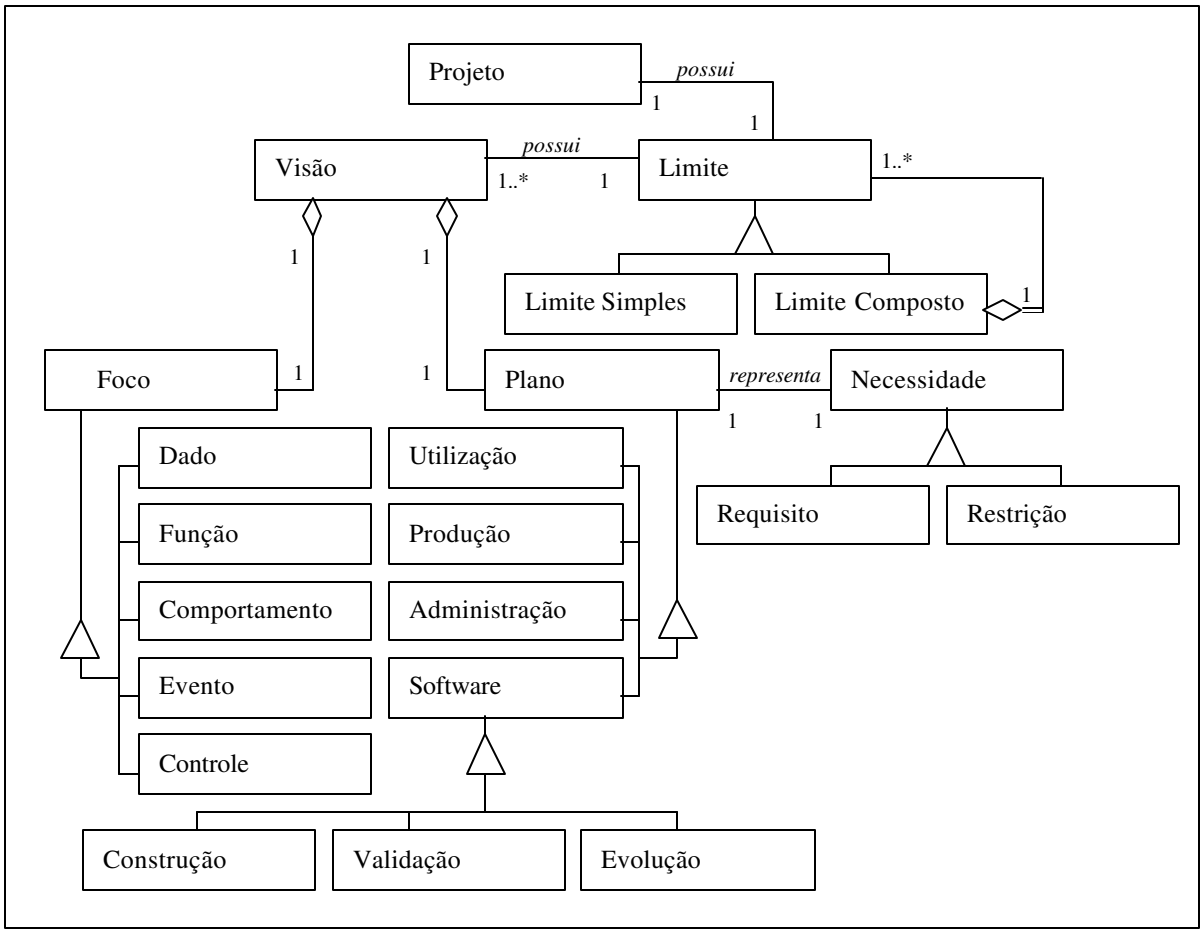


Figura 3.1 - Meta modelo da Análise por Visões

Este modelo representa as entidades, as classes, componentes da arquitetura referência, adaptada de [Picarelli02], denominada de *Análise por Visões*.

Segue-se primeiramente a caracterização da entidade *Projeto* e após as caracterizações das demais entidades do modelo.

Determinado um tipo de trabalho, no caso o desenvolvimento de um produto de software, escolhido um tema, uma área, um assunto e alocados os recursos necessários, já é o bastante para se definir um *Projeto* de uma aplicação de software. Um *Projeto* pode ter os atributos:

Projeto[1]:

Referência-Projeto

[Nome-Projeto]

[Nome-Fantasia-Projeto]*

[Resumo-Tema]

[Descrição-Tema]

[Data-prevista]*

[Data-real]*

[Observação]*

Os atributos *Referência-Projeto* e *Nome-Projeto* terão qualquer valor que identifique o projeto e são preenchidos de forma livre. Podem ser uma ou mais letras, números, letras e números ou uma ou mais siglas.

O atributo *Nome-Fantasia-Projeto* pode indicar uma referência ao projeto com teor comercial, de *marketing*, mercadológico ou motivacional.

Os atributos *Resumo-Tema, Descrição-Tema e Observação* são preenchidos, conforme habilidade do documentador, de maneira livre e em linguagem natural, na forma de representação do tipo *Descrição*.

As várias datas, que podem ser atribuídas por fases ou atividades, adjetivadas ainda com *de início* e *de término*, podem ser determinadas valorizando os atributos *Data-prevista* e *Data-real.*

[1] [ ]-opcionalidade; { }-alternabilidade; *-repetição; sem sinal-obrigatoriedade.Notação Gane (Gane79).

3.1. Entidade Limite

Ao se examinar um fenômeno determina-se arbitraria e imaginariamente o seu limite [Carvalho88]. O componente *Limite* é importante pois representa o alvo a ser atingido. A determinação correta deste alvo é *Fator Crítico de Sucesso - FCS* para um projeto de desenvolvimento de software [Feliciano88]. Após a determinação deste *Limite*, esse deverá ser revisado, testado e validado entre integrantes da equipe de desenvolvimento e comunidade usuária [Yourdon89][Champeaux93]. A idéia é garantir, de alguma forma, que o que será feito é realmente a coisa certa e é importante obter-se *feedback* dos *stakeholders* o mais cedo possível [Zhang02]. Descobrir logo possíveis defeitos reduz significativamente o custo de reparo necessário a fazer em uma aplicação de software [Faulk97].

Na determinação e especificação do *Limite* busca-se ter uma idéia do alvo, objetivo, abrangência, escopo, tamanho, domínio e complexidade do problema que está sendo tratado. Esta idéia deve ter um grau de precisão que possibilite, no mínimo, um primeiro esboço de orçamento e planejamento para o projeto.

A entidade *Limite*, como mostra o modelo na Figura 3.1, segue o padrão de projeto do tipo *Composite* [Gamma95]. Assim, desta forma modelada, já se possibilita um particionamento. Um limite pode ser particionado em outros, conforme conveniência de modelagem, e podem ser tratados de forma individual ou composta.

O componente *Limite* pode ser especificado como segue:

Limite:
Referência-Limite
Descrição-Limite
[Declaração-Objetivo]*
[Observação]*
Abordagem {Sistema, Objeto}

Estratégia: {*top-down*, *bottom-up*}
[Tipo-representação]: {Textual, Gráfica, Mista}
[Representação]*

O atributo *Referência-Limite* tem como valor a letra L seguida de um número inteiro que identifica o *Limite.*

Os atributos *Descrição-Limite* e *Declaração-Objetivo* são elaborados a partir dos atributos da entidade *Projeto.* A intenção principal é cercar o tema, esclarecer melhor, buscar um alvo mais claro, eliminar generalidades, buscar particularidades.

O atributo *Observação* é preenchido, conforme necessidade, também de maneira livre e em linguagem natural na forma de representação textual do tipo *Descrição*.

O atributo *Abordagem* indica a abstração escolhida para determinar e representar este *Limite.* Pode ser a abordagem sistêmica ou a orientação a objetos. Esta escolha não significa que o software será desenvolvido utilizando ou não a orientação a objetos. Significa apenas que a entidade *Limite* será determinada e especificada através de uma ou de outra abordagem.

O atributo *Estratégia* tem a ver com os tipos de operações mentais que são referentes a processos de refinamento e condensação de modelos [Bennaton84]. Podem ter o valores *top-down* ou *bottom-up* para estratégia que vai do geral para o particular e a que parte dos detalhes para o todo, respectivamente.

O atributo *Tipo-Representação* pode ter os valores *Textual*, *Gráfica* ou *Mista.*

O atributo *Representação* contém o produto da atividade de análise, no caso, os possíveis modelos gerados: modelos descritivos, diagramas, figuras, tabelas, textos e outros.

3.2. Entidade Visão

Para efeito de modelagem de requisitos, após determinação do *Limite*, o contexto referente ao projeto de desenvolvimento de um produto de software pode ser particionado em várias *visões*. A entidade *Visão* pode ter os atributos:

Visão:
Referência-Visão
Referência-Limite
Referência-Foco
Referência-Plano
[Observação]*

Os atributos *Referência-Visão, Referência-Limite, Referência-Foco e Referência-Plano* terão como conteúdo as letras *V, L, F* e *P*, respectivamente, seguidas de números inteiros que identificam a visão, limite, foco e plano.

O atributo *Observação*, é preenchido conforme a necessidade, também de maneira livre e em linguagem natural na forma de representação textual do tipo *Descrição*.

3.3. Entidade Foco

Componente de uma *Visão*, a entidade *Foco* representa aquilo que se quer dar destaque. Representa aquilo que se deseja dar maior importância e realçar no modelo. O componente *Foco* pode ser especificado como segue:

Foco:
Referência-Foco
Tipo-Foco: {Função, Dado, Comportamento, Controle, Evento, Outro}
Abordagem: {Sistema, Objeto}

Estratégia: {*top-down*, *bottom-up*}
[Observação]*
[Tipo-representação]: {Textual, Gráfica, Mista}
[Representação]*

O atributo *Referência-Foco* tem como valor a letra *F* seguida de um número inteiro que identifica o foco.

O atributo *Representação* contém o produto da atividade de análise, no caso, os possíveis modelos gerados: modelos descritivos, diagramas, figuras, tabelas, textos e outros.

O atributo *Tipo-Foco* indica quais são os aspectos principais que se pretendem especificar por vez. Não significa que esses aspectos se apresentem naturalmente de forma isolada, mas, que o modelador pretenda dar maior atenção, por vez, a uma determinada característica, dentro de um *Limite*. A lista de valores apresentada para *Tipo-Foco*: função, dado, comportamento, controle e evento, não se esgota. Pode, conforme necessidades e desejos do modelador, haver outros valores para tipo de *Foco*.

O *Tipo-Foco* de valor **Função** deve ser utilizado quando se deseja destacar uma ou mais ações que são executadas em um *Limite*. As ações fazem, executam algo.

O *Tipo-Foco* de valor **Dado** é utilizado quando se deseja destacar os diversos dados que existem, que são definidos e manipulados em um *Limite*. Os dados possuem uma direta associação semântica com o problema que está sendo tratado. Apesar do amplo leque de aparências, de mídia, de forma, de abstração e de contexto, um dado pode ser sempre considerado um *Elemento de Dados* ou uma *Estrutura de Dados*.

O *Tipo-Foco* de valor **Comportamento** pode ser utilizado quando se deseja destacar as características dinâmicas de um sistema, de um objeto ou de mais de um objeto agindo conjuntamente. Um comportamento atende a dois fatores: cumprem requisitos que

têm a ver com a passagem do tempo e desempenha funcionalidades embutidas, que são provocadas por estímulos externos.

O *Tipo-Foco* de valor **Controle** deve ser utilizado quando se deseja destacar o fluxo da passagem do controle entre as diversas funções de uma aplicação ou as condições especiais que levam a possíveis alternativas de ações operacionais. O controle é representado normalmente por um tipo de dado artificial, indicadores ou *flags*.

O *Tipo-Foco* de valor **Evento** deve ser utilizado quando se deseja destacar os eventos que influenciam o funcionamento de um sistema ou que provocam reações de um ou mais objetos. Um evento é qualquer ocorrência externa, instantânea, à entidade que está sendo estudada. Algo provocado por um agente externo. Evento significa sempre um fato ocorrido externamente à entidade alvo. Eventos são alterações no ambiente [Shiller90].

3.4. Entidade Plano

Os requisitos e restrições de um produto de software representam as características, atributos, propriedades, que devem ser contempladas por este produto [Sommerville97] [Rashid02]. Eles são determinados levando-se em conta a natureza do problema, os desejos, as necessidades, as prioridades e as idéias de clientes e usuários. Considera-se também as restrições tecnológicas, administrativas e econômicas, as oportunidades de negócios, as circunstâncias de desenvolvimento, as circunstâncias de operação do produto de software, os valores pessoais e as qualificações de seus desenvolvedores.

Esses requisitos e restrições aparecem de forma misturada através de processos de levantamentos, estudos e investigações de um problema ou de um contexto.

Como esses requisitos e restrições precisam ser tratados de formas diferentes e em momentos também diferentes, ter um critério de classificação adequado pode ser útil para organizá-los, avaliá-los, registrá-los, modelá-los e priorizá-los.

Vários critérios para esta classificação, como prioridade, objetivos, natureza da especificação e outros já foram propostos em [Lamsweerde00], [Silva01] e [Rashid02].

O componente de uma *Visão* de nome *Plano* é a proposta de classificação dos requisitos e restrições para ajudar neste tratamento. Ele representa, como mostra o modelo da Figura 3.1 as *Necessidade*s. Há modelos em que é tratado diferente *o que o usuário quer* e *o de que o usuário necessita* [Hatalsky02]. Neste modelo, essa caracterização será tratada como o atributo da entidade *Plano Tipo-Necessidade,* que poderá ter os valores *Requisito* ou *Restrição* e não *desejos* e *necessidades*.

O componente *Plano*, pode ser especificado como segue:

Plano:
Referência-Plano
Tipo-Necessidade: {Requisito, Restrição}
Natureza-Necessidade: {Funcional, Não Funcional}
Tipo-Plano: {Utilização, Produção, Administração, Software}

Para o conteúdo do atributo *Referência-Plano* pode ser utilizada a letra *P* seguida de um número inteiro que identifica o *Tipo-Plano*. Podem-se utilizar os inteiros 1, 2, 3 e 4 para as subclasses *Utilização, Produção, Administração e Software,* respectivamente.

Para o atributo *Tipo-Necessidade* escolhem-se os valores *Requisito* ou *Restrição*. Restrições são necessidades que devem ser cumpridas, porém, podem ser incluídas ou excluídas devido a fatores circunstanciais tais como: disponibilidade tecnológica, recursos alocados, investimentos efetuados e outros.

Para o atributo *Natureza-Necessidade* escolhem-se os valores clássicos, *Funcional* e *Não-Funcional*. Utilizam-se para caracterizar como funcionais requisitos do que a aplicação deve fazer e como os não-funcionais as necessidades de como esses requisitos devem ser implementados [Sommerville97].

O *Tipo-Plano* de valor **Utilização**, *Referência-Plano* de valor *P1*, trata dos requisitos clássicos. Quando se fala simplesmente em requisitos, refere-se a esse tipo de requisito. São os chamados requisitos da análise, requisitos lógicos, requisitos do *o que* fazer, requisitos essenciais, requisitos conceituais. São os requisitos isentos de tecnologia. São requisitos relativos à oferta, tratamento e uso das informações. É, portanto, o conjunto das informações oferecidas, informações armazenadas, necessidades e desejos da forma e do tratamento dessas informações, regras de negócios, regulamentos, normas, critérios de organização, legislação e estilos administrativos que, se contemplados, serão embutidos em uma aplicação de software.

No *Tipo-Plano* de valor **Produção**, *Referência-Plano* de valor *P2*, instanciam-se as necessidades relativas à geração do produto final, no caso a informação, quando a aplicação de software é colocada em operação.

Os requisitos desta categoria são relativos à tecnologia de hardware e software, aos procedimentos operacionais, a padrões de armazenamento e recuperação de dados e programas, ao controle de abastecimento de suprimentos, ao estabelecimento de ambientes operacionais de trabalho, às necessidades de implantação de sistemas de segurança física e lógica [Toigo89], à escala de prioridades de execução de programas, aos fatores éticos e legais de manipulação de dados, a seqüências necessárias de execução das aplicações, ao nível de serviço e de qualidade justificados, aos critérios de acesso, à criação e ao controle de bibliotecas de programas, aos esquemas de contingência e aos planos de *upgrades* tecnológicos adotados para gerar, cuidar e disponibilizar, para clientes e usuários, o principal produto: a informação.

Normalmente estabelecem-se três ambientes independentes: de desenvolvimento, de testes e de produção. Os fatores citados compõem o chamado *Ambiente de Produção*. Todas as aplicações de software devem estar de acordo com os requisitos estabelecidos por este ambiente de produção para possibilitar a sua operacionalização. Todas as aplicações de

software devem ser desenvolvidas, adaptadas, localizadas ou configuradas tendo como referência este ambiente de produção.

O *Tipo-Plano* de valor **Administração**, *Referência-Plano* de valor *P3*, contemplam as restrições gerenciais e administrativas do produto e do processo de construção da aplicação de software. Necessidades de prazos, limitações de recursos fazem parte desta categoria.

Os requisitos desta categoria são relativos, entre outras coisas, ao gerenciamento, acompanhamento e controle de um projeto de software. Gerenciamento que executa as atividades de planejar objetivos, organizar recursos, integrar esforços, medir o realizado e revisar o planejado [Jones85]. Projeto no sentido amplo da administração, que significa alocar recursos de uma organização, recursos esses que serão utilizados para alcançar um resultado qualquer [Chiavenato83]. Esses requisitos ou restrições são aqueles necessários para estabelecer valores, regras, alternativas, determinar estratégias, escolher técnicas, ferramentas, criar condições favoráveis, conseguir e disponibilizar recursos de toda ordem, que possibilitem o bom desempenho do processo de construção de uma aplicação de software.

O *Tipo-Plano* de valor **Software**, *Referência-Plano* de valor *P4*, tem como instâncias todas as características para a especificação e elaboração do produto de software.

Nesta subclasse, não são tratados requisitos ou restrições relativos ao produto informação, mas relativos à *máquina* que irá produzir a informação. Requisitos e restrições deste tipo são normalmente invisíveis para o cliente ou usuário final. Qualquer informação relativa às características do software em si fazem parte desta subclasse. Elas são baseadas em tecnologias disponíveis, métodos da *Engenharia de Software* e de valores e princípios da qualidade adotados pela equipe técnica. Adaptando-se o *processo de software* de Sommerville [Sommerville01], que possui as atividades de *especificação*, *desenvolvimento*, *validação* e *evolução* do software, esta classe é especializada em três subclasses:

. **Construção** de Software, que trata das necessidades referentes às atividades do fazer, do implementar. Atividades que embutem o risco de cometer defeitos.

. **Validação** de Software, que trata das necessidades referentes às atividades do verificar, inspecionar, testar e validar. Atividades que objetivam descobrir os defeitos.

. **Evolução** de Software, que trata das necessidades referentes às atividades de adaptar funcionalidades, de incluí-las e/ou excluí-las. São referentes também às atividades de remover os defeitos.

Seguem mais algumas considerações sobre *Limite, Foco e Plano*.

3.5. Considerações complementares - Limite, Foco e Plano

Nesta seção há comentários e considerações no sentido de orientar e recomendar alguns aspectos, em contraponto ao *o que* apresentado anteriormente, relativos agora ao *como* tratar *Limite, Foco e Plano*.

3.5.1. Limite

Em um projeto de construção de um software, tem por início o estabelecimento, a partir de um contexto escolhido, o(s) *Limite*(s) de um domínio específico.

Um *Limite* pode, dependendo do escopo, do tamanho e da complexidade do contexto, também ser particionado em outros *Limites*. Macro particionamento pode ser visto também no conceito de *Pacotes* na *UML* [Booch99] ou em *Assunto* [Coad91].

As respostas às seis perguntas conhecidas como *5W/1*H (*why, who, what, when, where, how*) podem ajudar no preenchimento dos atributos da entidade *Limite.*

Conceitos de missão, metas e objetivos [Goldratt90] [Pirsig84] e maneiras de determiná-los, apresentados em outros contextos tais como administração de empresas e *Planejamento Estratégico* [Gillenson84][Certo90] podem também ajudar na determinação dos *Limites*.

Pode-se elaborar um roteiro, através do raciocínio dedutivo, que, partindo-se de um tema central, chega às particularidades, ou pelo raciocínio indutivo, que, de idéias de unidades menores, chega às unidades maiores [Feitosa91]. Por exemplo, partindo-se do raciocínio indutivo para definir um conteúdo pode-se seguir os seguintes passos [Souther77]:

1 - fazer uma lista, de forma livre, de necessidades informacionais,
2 - definir um ponto central,
3 - fazer uma lista de idéias de forma livre *(brainstorming)* em torno deste centro,
4 - eliminar, após análise crítica, as idéias irrelevantes ou não pertinentes,
5 - categorizar e agrupar itens semelhantes,
6 - tentar denominar esses subgrupos,
7 - redefinir o ponto central.

Na *Engenharia de Requisitos* [Sommerville97] [Kotonya98], técnicas de elicitação de requisitos [Martins99] podem ser utilizadas para contribuir no cerco ao alvo que determina um *Limite*. Técnicas de *Análise Rápida* [Gane88b], *Análise de Discurso, Análise de Protocolo* [Ericsson84], *Prototipação* [Melendez90], técnicas orientadas a objetivos (*goal-oriented*) [Letier00] [Lamsweerde02][Weiss02], visam também a acertar e garantir o desenvolvimento certo da coisa certa.

Uma técnica orientada para equipes bem conhecida é a JAD [Chin95][Yourdon92]. Julga-se que, entre outras coisas, por adotar uma estratégia *top-down* e quebrar barreiras de comunicação, ela é mais produtiva do que um *brainstorming* [August91].

3.5.2. Foco

Tipo-Foco: Função

Recomenda-se que as funções sejam explicitadas, em qualquer contexto, através de um verbo mais substantivo. O verbo deve estar na forma ativa, como por exemplo *calcula*r ou *aprovar*, ou na terceira pessoa do singular do presente do indicativo, tal como *calcula* ou *aprova.* As funções podem ser modeladas em diversos níveis de detalhes. O atributo *Estratégia* oferece as opções de caminho para este detalhamento.

A especificação detalhada de uma função pode ser efetuada com maior ou menor grau de formalismo conforme necessidades e objetivos. O grau máximo de determinismo na especificação de uma função é conseguido através de sua representação em uma linguagem formal [Kovitz02]. Bastante utilizado, porém no contexto de desenvolvimento de *Sistemas de Informação*, é a especificação na categoria *DNL* (*Disciplined Natural Language)* [Smith02] tais como português estruturado, português compacto, árvores e tabelas de decisão [Gane79] [DeMarco89] [Pressman97].

Há métodos que fazem distinção para o *verbo mais substantivo.* Fazem distinção entre função, processo, serviço, módulo, operação, método, ação e atividade conforme o contexto e modelo onde o *verbo mais substantivo* aparece. Uma função pode ser, por exemplo, um *Case* no *Diagrama de Caso de Uso* [Jacobson92] ou uma *operação* no *Diagrama de Classes* [Rumbaugh91].

Propõe-se a classificação de funções em quatro tipos: transformação, qualificação, mista e de transformação de tecnologia. A de transformação altera totalmente o valor de um dado. A de qualificação altera o significado de um determinado dado, apenas adjetivando-o

de uma forma diferente. A mista transforma e qualifica parcialmente uma estrutura de dados. A de transformação de tecnologia realiza apenas modificação no formato físico de suporte da informação. Esta classificação pode ser útil na identificação, elaboração e validação de modelos funcionais e em atividades de mapeamento de processos. Normalmente, a função de transformação interage diretamente com agentes externos, gerando respostas. Função de qualificação ajuda a estabelecer e manter a memória do sistema.

A Figura 3.2 mostra um exemplo de cada tipo de função. O que está entre parênteses significa uma instância da estrutura ou do elemento de dado.

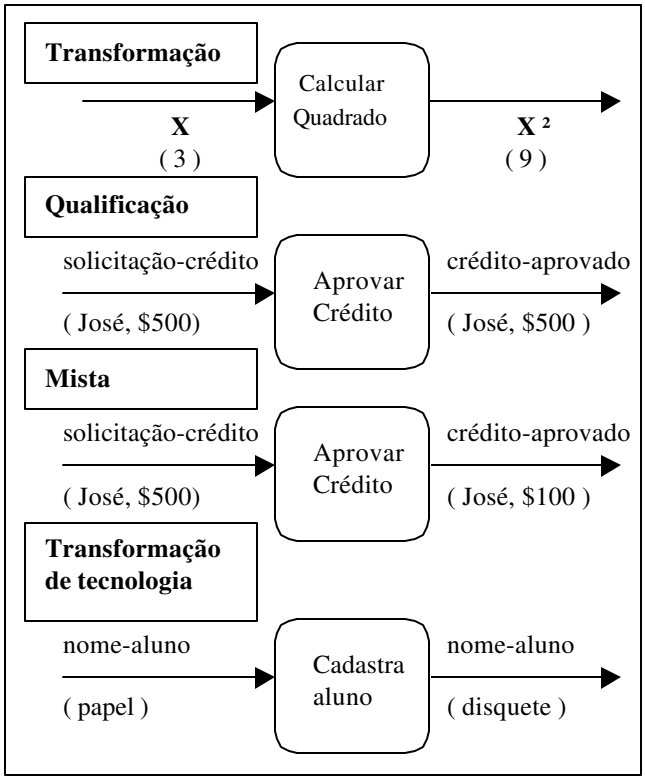


Figura 3.2 - Exemplo de cada tipo de função proposto

Tipo-Foco: Dado

Os dados são explicitados, em diversos contextos, através de um substantivo, de preferência no singular. Opcionalmente são seguidos de um ou mais adjetivos.

Identificar, definir, distribuir, classificar, modelar e organizar os dados em conjuntos separados que se podem relacionar, denomina-se como atividades de *Análise de Dados*. Ela teve repercussão no final da década de 70 na construção de aplicações orientadas a dados e suportadas por tecnologia de *Banco de Dados*, principalmente a de *Banco de Dados Relacionais* [Date84] [Korth86].

Os dados podem aparecer em um fluxo de dados, em um depósito de dados, em uma entidade, em uma tabela, em um arquivo, em um registro, em um atributo de uma classe, em um valor deste atributo, em um estado de um objeto, em um argumento de uma mensagem, em uma variável ou parâmetro de um programa, subprograma ou subrotina de computador.

Um dado pode ser apresentado em qualquer meio de entrada e saída, tal como um relatório, planilha ou tela de interação. Pode ser registrado em meios analógicos como papel, microficha, microfilme ou em qualquer meio de armazenamento digital. Pode ser representado na sua natureza multimídia da informação como texto, som, gráfico, imagem e imagem em pleno movimento [Wimblad90].

Os meta-dados, que representam os atributos dos dados, podem ser determinados e especificados em um *Dicionário de Dados*. Este *Dicionário de Dados* é uma ferramenta de software normalmente já integrada a um *DBMS*.

Tipo-Foco: Comportamento

Um comportamento pode ser especificado e apresentado, através de um conjunto de eventos e estados, em vários níveis de detalhes, conforme desejos e necessidades do modelador. Um estado pode ter subestados.

Para um estado o objeto ou o sistema possuem operações ou funções, normalmente chamadas de atividades, que são válidas de serem desempenhadas naquele estado específico.

A modelagem de uma sociedade de objetos, que é o comportamento que retrata um cenário quando agem juntos, pode ser efetuada graficamente por meio dos diagramas de interação o *DSQ - Diagrama de Seqüência e* o *DCO - Diagrama de Colaboração,* por exemplo. Usando-se o *DTE- Diagrama de Transição de Estados,* modela-se o comportamento individual de um objeto.

Algumas metodologias, *OMT* [Rumbaugh91], por exemplo, colocam a modelagem dinâmica, onde entra a modelagem do comportamento, nas etapas finais da análise de requisitos. Mellor [Mellor88] sugere que assim que um objeto é determinado sua modelagem comece pela representação de seus estados. Depois, para cada estado específico, levantem-se as operações válidas e representem-nas graficamente em um *DFD.*

Tipo-Foco: Controle

Os controles são explicitados através de dados que possuem, normalmente, valores binários, explicitado de forma codificada, indicando validade ou não validade, verdadeiro ou falso, certo ou errado, *bit* ligado ou desligado. No contexto de programação de computadores os controles são conhecidos como indicadores ou chaves. O termo em inglês *flag* também é utilizado. Normalmente se usam os verbos *ligar* ou *desligar* para denominar eventos que alteram o conteúdo de controles binários.

O uso de indicadores deve ser efetuado com certa parcimônia. Como eles aparecem codificados podem dificultar o entendimento de um programa. Para minimizar este problema recomenda-se que os indicadores, além da adoção de um padrão para nomeá-los, também sejam dicionarizados como *Elemento de dado* que tenham os atributos *Nome-indicador , Valor-código(s) e Significado(s).*

Outro sentido dado ao termo controle é o de monitorar e acompanhar um fluxo de funções que são executadas sob algumas condições e em determinada seqüência.

Tipo-Foco: Evento

Eleger um conjunto de eventos, estabelecer suas seqüências, condições para suas ocorrências, reações pré-definidas da aplicação que eles influenciam denomina-se análise ou programação, dirigidas por eventos. Por exemplo, uma linguagem de programação [Sebesta99], um software de autoria, ou uma aplicação qualquer de software pode ser denominada como dirigidas por eventos (*event driven)* [Nunes00] [Brown00].

Considerações específicas sobre eventos são tratadas na *Análise de Eventos* (seção 3.6).

3.5.3. Plano

Assim que os desejos e as necessidades que uma aplicação de software deve contemplar e vão sendo explicitados, o modelador pode interpretá-los, caracterizá-los e agrupá-los conforme as características de cada classe de *Plano* do modelo apresentado na Figura 3.1.

Tipo-Plano: Utilização

Relativamente aos outros tipos, esses requisitos têm uma vida mais longa. São mais perenes do que, por exemplo, os relativos à tecnologia. São os requisitos que determinam o que a aplicação de software deverá oferecer em termos de informações.

As informações oferecidas por uma aplicação vão apoiar processos administrativos ou produtivos de uma organização. Esses processos podem ser distribuídos em uma pirâmide [Luporini85], dividida em três faixas horizontais. A faixa superior representa os processos estratégicos, a do meio os processos táticos e a faixa da base os processos operacionais [Feliciano96].

Algumas aplicações de software têm, baseadas nesta pirâmide [Yourdon89], classificação semelhantes tais como *MIS - Management Information Sytems; DSS - Decision Support Systems; EIS - Executive Information Systems* [Furlan94]; *ERP - Enterprise Resource Planning* [Haberkorn99] e outros.

Daí dizer, inadequadamente, informação estratégica, informação tática ou informação operacional. A informação em si não é estratégica, tática nem operacional. O processo que ela suporta é que pode ser assim categorizado como operacional, tático ou estratégico.

Indo de baixo para cima nesta pirâmide, as necessidades informacionais requerem periodicidade, previsibilidade e detalhes de maior para um menor grau. De forma bem resumida, os processos operacionais são relativos ao presente. Eles são tocados por técnicos e por especialistas. Os processos táticos são relativos ao passado, ao acompanhamento e controle e são tocados pela média gerência. Os processos estratégicos são relativos ao futuro, às decisões de alto risco, conduzidos pela alta gerência, diretorias e presidentes das organizações.

A principal fonte de requisitos deste tipo são os usuários e clientes. Eles, por criarem, operarem e atuarem nos sistemas são os especialistas do domínio do negócio [Furlan97]. Embora existam inúmeras razões para não colaborarem, é fundamental, porém, sua ajuda [Kiedaisch01]. O conhecimento que eles possuem precisam ser transferidos para os desenvolvedores de software, os analistas de sistemas ou de negócios [Saviani92]. Essa transferência deve ser medida [Vickers02] e efetuada de forma planejada e organizada. Esses conhecimentos são registrados, inicialmente, em linguagem natural, tendo assim alto potencial de inconsistências e ambigüidades que precisam ser removidas [Zowghi01].

Tipo-Plano: Produção

Estabelecer e manter atualizado um ambiente de produção, no largo espectro que vai da computação pessoal à computação corporativa, não é tarefa das mais fáceis [Walton89]. Isto é normalmente efetuado por consultores, especialistas autônomos, empresas especializadas, por comissões especiais ou comitês montados na própria organização. Esses últimos são geralmente denominados de *Comitês de Informática* [Umbaugh89] e compostos por profissionais selecionados do quadro de empregados das próprias organizações onde o ambiente de produção será instalado e mantido. Esses comitês têm um caráter executivo, consultivo ou misto. Eles executam, supervisionam ou simplesmente estabelecem diretrizes e prioridades para a aquisição e implantação da tecnologia da informação. Estabelecem e mantêm na empresa o seu parque computacional. Isto é efetuado sob um plano conhecido pela sigla *PDI - Plano Diretor de Informática* [Haberkorn99]. Este plano tem abrangência de dois a cinco anos, com revisão e atualização anual. Ele pode contemplar um leque que vai do hardware, software de infra-estrutura, os software básicos, os recursos humanos até as aplicações de software de alta plataforma.

Em um contexto de mudanças diárias em que a informática está inserida, é importante realçar que este ambiente de produção precisa de certa estabilidade. Quanto mais estável for, melhor será considerado. As pessoas, usuárias desta tecnologia, chegam ao trabalho e não podem ser surpreendidas por um ambiente diferente. Cada *upgrade* tecnológico deve ser rigorosamente planejado e divulgado. Um bom esquema de treinamento deve existir como suporte das mudanças necessárias, para não comprometer o grau de produtividade dos profissionais usuários da tecnologia de informação [Jones86].

Os requisitos desta classe, os requisitos de produção, não podem ser esquecidos mesmo no âmbito da computação pessoal. Uma aplicação de software é às vezes instalada, por exemplo, sem que se estabeleça bibliotecas e nomes padrões para o armazenamento de arquivos. Procedimentos e responsabilidades operacionais básicas, tais como disponibilizar diariamente a rede ou esquema de reposição de papel nas impressoras, não são idealizados nem estabelecidos. Com o passar do tempo, o grau de organização pode ficar tão baixo, a

recuperação das informações pode ficar tão difícil e o índice de retrabalho tão alto, que pode acabar dando às pessoas a sensação de que a informática mais atrapalha do que ajuda.

É preciso adicionar, levantar e representar, esses tipos de requisitos ou restrições, requisitos não funcionais, sobre os modelos dos requisitos funcionais. Em aplicações onde o desempenho representa fator crítico isto não é tarefa fácil [Woodside02]. Porém, no contexto de aplicações de alto nível a proposta é determinar os requisitos básicos de produção através dos meios físicos clássicos. Esses meios são os dispositivos de armazenamento e de entrada e/ou saída. São os arquivos, os bancos de dados, disquetes, discos rígidos, CDs, relatórios e planilhas em papel, microfilmes e microfichas, hiperdocumentos, *homepages,* telas e janelas interativas de consulta ou de entrada de dados.

O levantamento dos requisitos mínimos de *Produção* pode ser efetuado utilizando-se como referência a lista de itens propostos conforme o meio físico.

Para o meio físico arquivo podem-se ter os requisitos básicos de produção:

- nomes padrões para os membros e para as bibliotecas de dados e programas.
- identificação externa do dispositivo por meio de etiquetas apropriadas.
- critérios para retenção de um arquivo: por versões, por prazo ou por data fixa.
- esquemas e procedimentos para fazer as cópias de segurança.
- local de armazenamento.
- estoque e reserva de mídias virgens.

Para o meio físico relatórios, planilhas em papel, microfichas podem-se ter os requisitos básicos de produção:

- códigos padrões para referências e controle de versões.
- títulos e *labels* de campos.
- datas de emissão, de referência e de validade.
- paginação.

- páginas rostos para a seleção e separação conforme quebra necessária.
- critérios para avaliação da qualidade de impressão.
- quantidade de cópias e destino das vias.
- empastamento, manipulação e transporte.
- estoque e reserva de papel e filmes.
- estoque e reserva de tinta, *tonner*, química fotográfica.
- critérios para destino após uso.
- processo de destruição, incineração, confecção de rascunhos, reciclagem.
- processo de digitalização, digitação, leitura ótica, *scann*er.

Para o meio físico janelas e telas interativas podem-se ter os requisitos básicos de produção:

- códigos padrões para referências e controle de versões.
- título e *labels* de campos.
- datas e horas de emissão, de referência e de validade.
- padrões de teclas e botões.
- padrão de navegabilidade e de interface.
- tempo de resposta e tempo de *time-out.*
- níveis de acesso.
- processo de cadastramento, habilitação e desabilitação de usuários.

Tipo de Plano: Administração

Nesta subclasse, são contempladas as necessidades para a elaboração do processo, planejamento, gestão e controle do projeto de desenvolvimento de uma aplicação de software; são levantados os recursos disponíveis, elaborados os cronogramas; são escolhidas as técnicas, métodos e ferramentas de apoio; são estabelecidos os pontos de controle; são adotadas as métricas [Whitmire95] e são acordados os esquemas de trabalho. As características organizacionais para tratamento da tecnologia de informação, para

desenvolvimento e implantação de produtos de software, dentro de uma organização qualquer, devem ser consideradas. Há alternativas propostas por Bosch [Bosch01].

Um modelo simples que mostra os fatores que influenciam no comportamento de um processo é o *Diagrama de Causas e Efeitos*, diagrama de *Ishikawa* [Ross88], apresentado na Figura 3.3.

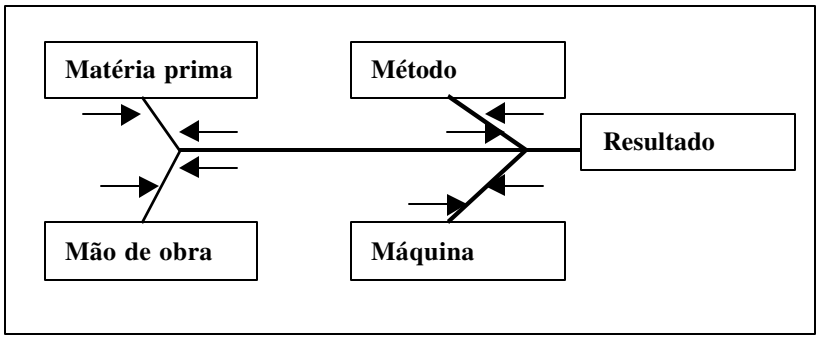


Figura 3.3 - Diagrama de Causa e Efeito (Ishikawa)

Este diagrama simples pode ser de grande utilidade no levantamento dos requisitos/restrições da subclasse *Administração*. A razão de sua escolha é justamente por que a sua simplicidade agrega um alto potencial de aplicação prática.

É proposto que, para cada projeto de desenvolvimento de uma aplicação de software, tendo como referências o diagrama e as considerações também propostas, efetuem-se o levantamento dos requisitos administrativos.

Considerações sobre a aplicação do Diagrama de Causa e Efeito estão descritas na seção 3.8.

Tipo de Plano: Software - Construção

Os requisitos ou restrições não são independentes uns dos outros [Giesen02]. Os requisitos/restrições pertencentes à subclasse *Construção de Software*, são obtidos na resposta do *como* implementar o *o que* já obtido. Assim, tem como matéria prima as necessidades já levantadas nas outras subclasses:

- Utilização: de onde se obtêm as informações a serem oferecidas.
- Produção: de onde se obtém o ambiente tecnológico operacional adotado.
- Administração: de onde se obtêm as diretrizes de escolha de fornecimento de produtos de software.

A partir das informações que serão tratadas e oferecidas pela aplicação, especificadas na subclasse *Utilização*, buscam-se aqui os requisitos inerentes a uma solução de software para tratamento, para a geração dessas informações. São os requisitos denominados no *CMM - Capability Maturity Model* de requisitos *alocados* [Fiorini98].

A técnica ou roteiro proposto para levantamento dos requisitos de *Construção de software* é, a partir das informações oferecidas, ou seja determinar quais informações devem ser armazenadas e quais devem ser geradas por um módulo de software.

Simbolicamente isto pode ser explicitado pela *fórmula* proposta **IO = IA + IP**, onde:

. "**IO**" são "**I**nformações **O**ferecidas",
. "**IA"** são "**I**nformações **A**rmazenadas",
. "**IP"** são "**I**nformações **P**rocessadas".

As informações *Armazenadas* são as capturadas, mantidas e disponibilizadas através de algum método de acesso. As informações *Processadas* são as obtidas através de um ou mais algoritmos. Assim, para se oferecer uma determinada informação, pode-se apenas

acessá-la e disponibilizá-la ou, pode existir a necessidade da execução de um ou mais programas para construir a informação oferecida.

A decisão entre armazenar ou gerar por processo pode ser tomada baseando-se em dois aspectos principais:

1. Grau de determinismo de geração da informação,
2. Estabilidade da informação.

O primeiro aspecto, grau de determinismo de geração de certa informação, avalia o quanto de acerto um algoritmo matemático, levando em consideração seu grau de complexidade para futura implantação, consegue, a partir de algumas informações, derivar a outra.

Alguns fatores culturais, legais ou éticos podem impedir o armazenamento de algumas informações obrigando, se ainda permanece a necessidade de oferecê-las, a gerá-las através de processos sofisticados de derivação.

O segundo aspecto, grau de estabilidade da informação, avalia a potencialidade de mudança da informação no decorrer do tempo. Dados históricos, por exemplo, possuem alto grau de estabilidade, pois, o passado não se altera.

Tendo-se determinado quais informações serão oferecidas, quais serão armazenadas e quais informações serão geradas por processo, a proposta é determinar quais os módulos de software, os componentes, objetos, programas, rotinas e subrotinas, necessários para armazenar, processar e oferecer as informações requisitadas.

Isto depende diretamente dos requisitos/restrições da classe *Produção* relativos à tecnologia de hardware e software em que a aplicação de software irá ser operacionalizada. Este conjunto de tecnologia instalado e disponível é chamado, de forma geral, de ambiente de produção. Assim, dependendo do ambiente de produção existente projetam-se os componentes de software necessários para a aplicação específica.

Informar é destruir incertezas em um determinado formato [Ward86]. Esta questão tem a ver justamente com o formato, com a tecnologia adotada. Tendo-se as informações oferecidas, armazenadas e processadas, a tecnologia disponível e as diretrizes de projeto de software, determinam-se então os componentes de software necessários para a aplicação.

É apresentado, neste trabalho uma proposta denominada *Análise de Componentes* (seção 3.7) para auxiliar a determinação dos componentes de software necessários.

Tipo de Plano: Software - Validação

Os requisitos/restrições pertencentes à subclasse *Validação de Software* são aqueles que, alocados ao software, possibilitam e facilitam a execução de processos e tarefas de revisão, validação, verificação, inspeção, teste, simulação e aceitação de produtos de software.

As próprias técnicas e métodos disponíveis pela *Engenharia de Software* das tarefas acima citadas, tais como teste de integração [Harrold91], testes baseado em fluxo de dados [Ntafos88], geração automática de testes [Korel90] [Weyuker86], testes de regressão, revisões estruturadas [Yourdon85] [Nogueira87], JAD [August91], de consistência entre especificação e requisitos [Kozlenkov02] e outras, induzem, de forma geral, no conjunto de características que os subprodutos de software da aplicação devem contemplar para ter facilidade de teste, verificação e validação.

Outra possível fonte de demanda de requisitos deste tipo pode ser a equipe que irá participar na execução das atividades de verificação, testes e aceitação dos subprodutos. Assim, recomenda-se que esta equipe, normalmente chamada de *equipe de testes*, quando existir, seja tratada como um tipo de usuário, pois também é geradora de requisitos/ restrições [Demarco82] da aplicação de software.

Tipo de Plano: Software - Evolução

Uma aplicação de software, logo que implementada, terá necessidade de mudanças. Os usuários irão solicitar muitas mudanças durante o período operacional da aplicação [Heales00]. A viabilização de produção de uma nova versão deve ser esperada. São pertencentes à subclasse *Evolução de Software* os requisitos/restrições que devem ser alocados ao software para possibilitar as modificações com uma segurança máxima e um custo mínimo.

As necessidades de mudanças vêm da comunidade usuária, da própria equipe técnica e da necessidade de remoção de defeitos. Nas atividades de remoção de defeitos, não raro, há a inserção involuntária de outros. Cada solução é fonte do problema seguinte [Gause90]. Os requisitos/restrições que caracterizam a facilidade de modificação são os requisitos da subclasse evolução de software.

3.6. Análise dos Eventos

A proposta da *Análise dos Eventos* é o resultado de uma adaptação da chamada *Análise Essencial* [McMenamin84] e da *Análise Estruturada Moderna* [Yourdon89]. Ela é apresentada em três passos:

1. Caracterização dos Eventos,
2. Geração da Lista de Eventos e
3. Reação a Eventos.

Passo 1. Caracterização dos Eventos

Propõe-se que a representação textual de um evento seja uma frase completa. O sujeito representa o agente externo, provocador do evento. Em seqüência, um ou mais, verbos significam as ações executadas por este agente e, logo em seguida, os objetos complementam a representação textual de um evento.

Pode-se ter uma frase explicitando um fato qualquer. Esta ocorrência pode ser considerada um evento para um sistema se ela não fizer parte das funções executadas por este sistema. Pode ser considerada um evento para um objeto se ela não fizer parte do conjunto de operações deste objeto. Um evento é sempre uma ocorrência externa à entidade observada.

Classifica-se o evento conforme o sujeito que representa o agente provocador. Quando o agente provocador for um objeto, pessoa, sistema, seção, setor, divisão, departamento, que representa um papel externo qualquer, classifica-se o evento como um *Evento Externo* [McMenamin84]. Quando o sujeito representa algo relacionado com o tempo cronológico o evento é dito como *Evento Temporal* [McMenamin84]. O tempo específico não é explicitado na declaração de um evento temporal.

Para eventos temporais em que o tempo for imprevisível o evento é denominado de *Evento Temporal de Controle*, ou simplesmente *Evento de Controle* [Yourdon89]. A razão para este nome é que para o evento ser percebido é necessária a emissão de um controle, que pode ser representado por um fluxo de controle, indicando sua ocorrência. Se, para explicitar sua ocorrência, algo externo deve ordenar, através de um indicador, classifica-se então este evento de *Evento de Controle*.

É importante realçar que o tempo sempre é um agente externo. O tempo, por princípio, faz parte do ambiente de todo sistema. Nenhum sistema tem controle sobre o tempo cronológico. Assim, qualquer provocação realizada por um agente relativo ao tempo é, de fato, um evento.

Um evento especial, de natureza temporal utilizado em várias aplicações de software, é o evento *idle*. Ele representa a não ocorrência de eventos durante algum tempo. Dá a idéia de vazio, de nada, de ocioso, de nenhuma ocorrência. Este evento é especial pois, não é declarado na forma sujeito/verbo/objeto. Se nenhum evento ocorreu durante um tempo pré-determinado, é disparado o estímulo relativo a este evento. Para que este tempo

seja novamente reiniciado pode-se ter duas alternativas: através de uma ordem externa, representada por um fluxo de controle, ou, através de outro evento temporal.

A Tabela 3.1 mostra a proposta de uma classificação, baseada no agente provocador, mais detalhada para os eventos temporais. Esta classificação pode facilitar a futura modelagem da reação do sistema na ocorrência de um evento. O evento tipo *1* é o *Evento Externo* e os demais, de *2* a *5* uma especialização dos evento temporais.

Tabela 3.1 - Tipos de eventos

Agente Provocador	**Tipo Evento**	**Exemplos**
Explícito Planejado	1	.Cliente compra produto .*Double click* no botão esquerdo no *mouse*
Explícito *ad-hoc*	2	.Gerente solicita emissão cobrança devedores .Técnico autoriza utilização equipamento
Tempo Imprevisível	3	.Momento de aplicar injeção .Momento de abastecer papel mecanismo impressor
Tempo Previsível	4	.Momento de efetuar manutenção veículo .Momento de solicitar mudança de senha
Ocioso	5	.Após 5 minutos (disparar o *screen-save*) .Quando atingir valor 15 (contatar rede)

Passo 2. Geração da Lista de Eventos

Um evento pode ser analisado. Usa-se o termo análise tanto para significar a análise em si mas também o trabalho inverso, no caso a síntese. A análise de um evento, tem aqui o significado de particioná-los. Síntese tem o significado de agrupá-los. A análise, ou síntese, é efetuada utilizando-se a estratégia denominada de *top-down*, do geral para o particular, ou de maneira inversa, denominada *bottom-up*. Ela é escolhida conforme o evento se apresente originalmente. Ele pode apresentar-se de forma sintética ou de forma analítica. Esta análise dos eventos é útil para determinar se diversos eventos podem ser agrupados, resumidos em um só, ou se um evento que é descoberto da forma sintética, deve ou não ser particionado em outros.

Descoberto um evento, tenta-se generalizar ou especializar suas partes, no intuito de gerar novos eventos, a partir deste. Recomenda-se que a análise de um evento seja efetuada na ordem:

1 - Objeto
2 - Sujeito
3 - Verbo

Não há uma razão técnica para esta ordem. Simplesmente deixa-se o trabalho de especialização ou generalização do verbo por último, por julgar-se, baseado em experiências práticas, ser este trabalho menos comum do que especializar ou generalizar substantivos.

Assim, se o evento apresenta-se originalmente de forma sintética, fixam-se primeiramente o sujeito e o verbo e escreve-se o evento para tantos quantos forem os objetos especializados. Depois, fixam-se os objetos e o verbo e especializa-se o sujeito. Gera-se daí uma nova lista de eventos. Por último, fixam-se os objetos e o sujeito fazendo variações para o(s) verbo(s), buscando outros verbos, mas de forma que a frase continue fazendo sentido para o sujeito e para os objetos daquele evento.

Não há uma maneira determinística para a realização dessas tarefas. Elas podem ser efetuadas de forma livre sem pré-crítica ou pré-censura, no âmbito individual, ou em equipe, através de técnicas de *brainstorms.* O objetivo é gerar uma lista grande de eventos para que não ocorra o risco de esquecimento de algum.

O inverso, ou seja, efetuar generalizações a partir de uma lista de eventos também pode ser realizada na mesma ordem citada acima, selecionando-se, por vez, sempre duplas de eventos, mas com ressalva. Só se deve generalizar, ou seja, de dois eventos encontrados escrever apenas um genérico, após se ter como resposta um *não* para cada uma das perguntas:

1 - A aplicação pretende reagir a este evento?

2 - A aplicação reage de forma diferente a esses dois eventos?

Para um *não* como resposta à primeira pergunta, pode-se desprezar simplesmente este evento. Para um *sim* como resposta à primeira pergunta e um *não* à segunda, pode-se, como foi dito anteriormente, juntar estes dois eventos em um evento genérico. Para *sim* como resposta às duas perguntas deve-se deixar a dupla de eventos como sendo dois eventos diferentes.

No primeiro passo deve-se eliminar os eventos que sabiamente a aplicação não contempla através de qualquer reação.

Com a lista de eventos que restou combinam-se os elementos de cada evento entre eles, gerando novos eventos. É preciso que se tenha o cuidado de não gerar eventos absurdos que foram explicitados através de frases sem sentidos. Obtida esta nova relação pode-se executar novamente o passo de eliminação de eventos e obter-se a lista final.

Escolhida, por exemplo, uma aplicação de *Vendas de Produto*, pode-se ter descoberto inicialmente o evento:

Cliente solicita compra de produto.

Como o evento apresenta-se de forma sintética, e, o *Cliente* e o *produto* apresentam-se de forma genérica, pode-se especializá-los seguindo a recomendação citada anteriormente.

1 - Especializando o(s) objeto(s):

1.1 - Cliente solicita compra de *sorvete.*

1.2 - Cliente solicita compra de *livro*.

1.3 - Cliente solicita compra de *arma*.

1.4 - Cliente solicita compra de *remédio*.

1.5 - Cliente solicita compra de *seguro de vida*.

2- Especializando o sujeito

2.1 - *Governo* solicita compra de produto.

2.2 - *Empresa* solicita compra de produto.

2.3 - *Pessoa* solicita compra de produto.

2.4 - *Criança* solicita compra de produto.

3 - Variando o verbo

3.1 - Cliente	*solicita compra*	de produto.
3.2 - Cliente	*solicita aluguel*	de produto.
3.3 - Cliente	*solicita disponibilidade*	de produto.
3.4 - Cliente	*solicita devolução*	de produto.

4 - Combinando eventos

4.1 - Criança	solicita compra	de arma.
4.2 - Criança	solicita devolução	de sorvete.
4.3 - Criança	solicita compra	de remédio.
4.4 - Pessoa	solicita aluguel	de remédio.
4.5 - Pessoa	solicita devolução	de livro.

5 - Eliminando eventos

Supondo que a aplicação não possui nenhum produto que possa ser alugado, pode-se eliminar, por exemplo, o evento *3.2*. Se a aplicação exigisse que para atender a um cliente este deva possuir inscrição como contribuinte da Receita Federal ou algo do tipo, pode-se eliminar o evento *2.4*. Em se tratando de um comércio não marginal, os eventos

4.1 e *4.3*, poderiam ser também eliminados. Os eventos *4.2* e *4.4* podem ser eliminados pois representam um absurdo.

Assim, após a eliminação de alguns eventos da lista tem-se uma nova lista. Repete-se esta análise até que se obtenha a relação final dos eventos que serão tratados pela aplicação alvo.

Passo 3. Reação a Eventos

Na *Análise Essencial* [McMenamin84] e na *Análise Estruturada Moderna* [Yourdon89], propõe-se que a modelagem da reação a um evento seja efetuada através de um *DFD***.** O diagrama aqui proposto, o *DRE - Diagrama de Reação a Eventos* é um tipo de *DFD* pois também se utiliza dos elementos fluxo de dados, depósito e função, porém difere em vários pontos. Primeiramente em seu objetivo, é um *DFD* específico para modelar a reação de eventos. Segundo, diferencia graficamente o fluxo de dados do fluxo de controle, a função fundamental da custodial, o agente externo dos que exercem atividades *ad-hoc,* a memória do sistema e os estados dos objetos da memória. Terceiro, apresenta um leque maior de possíveis reações e, por último, possui regras próprias para sua elaboração e avaliação.

As funções que exercem atividades ligadas à reação do evento, à missão principal e aos objetivos do sistemas são chamadas de atividades fundamentais. As que exercem atividades de apoio e suporte às atividades fundamentais são chamadas de atividades custodiais. As atividades custodiais exercem, normalmente, funções de qualificação. Para qualquer sistema, as atividades custodiais são fartas e as atividades fundamentais são raras.

Propõe-se representação gráfica diferente para as fundamentais e custodiais. No *DRE* proposto as atividades fundamentais são representadas por um círculo e as atividades custodiais são representadas por uma elipse. Representar essas atividades de forma separada ou juntas, implicitamente as custodiais como parte interna das fundamentais, fica a critério do modelador.

É importante realçar que uma atividade custodial tem que ter obrigatoriamente, por convenção, para ser referente à reação do mesmo evento, uma ligação com a atividade fundamental através de um fluxo de dados ou de controle. Atividades fundamentais ou custodiais que aparecem em um modelo sem ligações entre elas, ou seja, de forma isolada, compartilhando ou não uma mesma memória, constitui um erro de modelagem. Este fato, ausência de ligação, significa que essas atividades isoladas pertencem à reação de outro evento e não o que está sendo modelado.

A memória do sistema, como na Análise Essencial, pode ser particionada em diversas entidades, sendo ou não consideradas, cada uma, uma classe de objetos. No *DRE* elas são representadas por um retângulo, com sua maior dimensão na horizontal subdividido em duas partes. Na parte superior, coloca-se internamente o nome da entidade, um substantivo no caso. Na parte inferior, subdivide-se caso haja, através de linhas verticais, em tantos setores quantos forem os estados desta entidade. Colocam-se nesses setores as referências numéricas e inteiras dos possíveis estados do objeto.

O estímulo de um evento pode ser representado graficamente através de um fluxo no sentido *agente provocador* para a aplicação. Para os eventos do tipo *1*, seu estímulo pode ser representado por um fluxo de dados representado por segmento de reta cheio. Para os demais tipos de eventos, por um fluxo de controle representado por segmento de reta pontilhado.

O *DRE* possui então os seguintes elementos com sua representação gráfica:

- Fluxo de dados: segmentos de retas cheios.
- Fluxo de Controle: segmentos de retas pontilhados.
- Atividade Fundamental: círculo.
- Atividade Custodial: elipse.
- Agente Explícito planejado: retângulo.
- Agente Explícito *ad hoc* e Tempo Imprevisível: retângulo com dois setores.

- Tempo Previsível, Ocioso: não se representa graficamente.
- Memória: retângulo subdivido em dois setores.
- Estados: setores subdividindo o setor inferior da memória.

A Figura 3.4 mostra, de forma genérica um *DRE*.

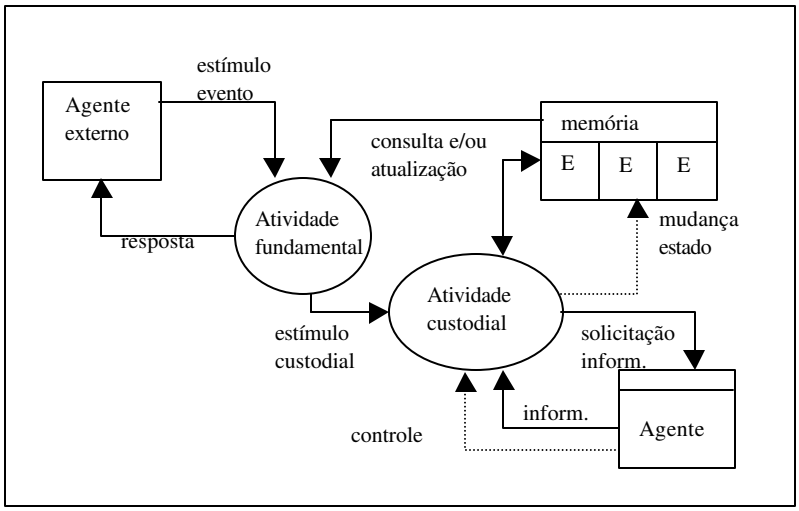


Figura 3.4 - DRE - Diagrama de Reação a Eventos

As reações de um sistema são as respostas *sim* às perguntas de número *03* e/ou *05* e /ou *07* originais da *Análise Essencial.* As outras perguntas aqui apresentadas são para efeito de facilitar a modelagem das reações a um evento. Temos então as oito perguntas que seguem:

01- Há necessidade de um Fluxo de Dados representando o estímulo do evento?
02- Há necessidade de um Fluxo de Controle representando o estímulo do evento?
03- Há uma ou mais respostas ao ambiente?
04- Há um estímulo da atividade fundamental para uma atividade custodial?
05- Há mudança de estado de um ou mais objetos?
06- Há necessidade de consulta à memória do sistema?
07- Há necessidade de atualização da memória do sistema?
08- Há necessidade de pedir/receber informações ao agente *ad-hoc* do sistema?

Para todo evento do tipo *1*, evento externo, é necessário um Fluxo de dados representando o estímulo do evento, reação de número *01*. Esse fluxo tem origem em um retângulo e o destino em um círculo.

Para eventos dos tipos *2* e *3* são necessários Fluxos de Controle para tal representação, reação de número *02*. Esses fluxos têm origem em um quadrado e o destino em um círculo ou em uma elipse.

A reação de número *03* é efetuada pela atividade fundamental e pode ser representada através de um ou mais fluxos de dados com origem em um círculo e o destino em um ou mais retângulos, os agentes externos.

A reação de número *04* é representada por um fluxo de dados ou de controle com origem no círculo, atividade fundamental, e o destino em uma ou mais elipses, atividade(s) custodial(ais).

A reação de número *05* é representada por um fluxo de controle, sem nome, com origem no círculo ou na elipse, atividade fundamental ou custodial, e o destino no novo estado do objeto, setor inferior do retângulo que representa o estado na memória.

A reação de número *06* é representada por um fluxo de dados com origem na memória e o destino na atividade fundamental ou custodial.

A reação de número *07* é representada por um fluxo de dados com origem na atividade fundamental ou custodial e o destino na memória.

A reação de número *08* é representada por um fluxo de dados com origem na atividade fundamental ou custodial e o destino no agente *ad-hoc* e por um fluxo de dados com sentido inverso, ou seja, do agente *ad-hoc* para a atividade.

Assim, se modelada separadamente, as atividades custodiais podem ter como entrada:

- obrigatoriamente um estímulo advindo da atividade fundamental que pode ser representado por um fluxo de dados ou por um fluxo de controle e/ou,
- uma consulta à memória do sistema, representada por um fluxo de dados e/ou,
- uma recepção de informações de um agente externo que exerce atividade *ad-hoc*, representado por um fluxo de dados,

As atividades custodiais podem ter como saída:

- uma solicitação de informações para um agente explícito *ad-hoc*, representado por um fluxo de dados e/ou,
- uma atualização na memória do sistema, representado por um fluxo de dados.

A Figura 3.5 mostra um exemplo de um modelo de reação ao evento do tipo *1*, *Pessoa deseja significado de palavra.*

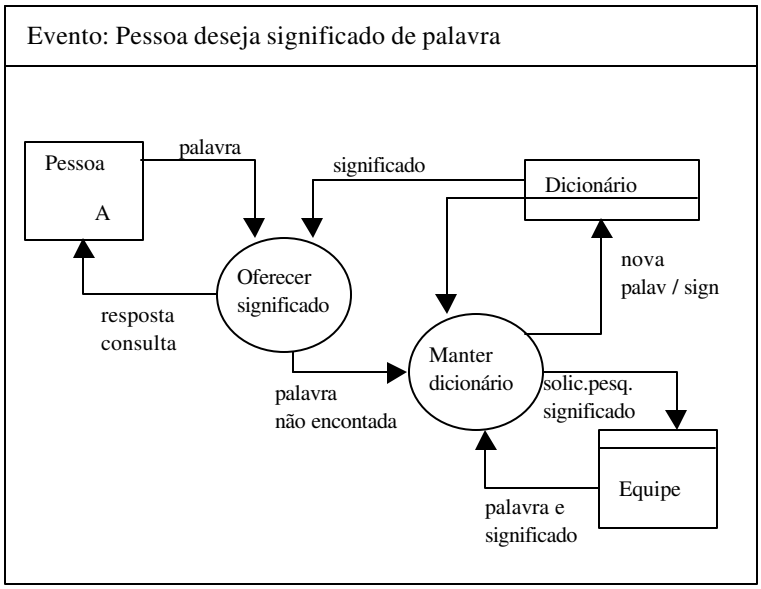


Figura 3.5 - DRE - Pessoa deseja significado de palavra

Para esta modelagem, as oito perguntas foram respondidas como segue:

01- Sim.

02- Não.

03- Sim.

04- Sim.

05- Não.

06- Sim.

07- Sim.

08- Sim.

Neste exemplo, as atividades fundamentais e custodiais foram modeladas separadamente. A atividade custodial, *Manter e Estabelecer Dicionário* aparece separadamente da atividade fundamental *Oferecer Significado de Palavras.*

3.7. Análise de Componentes

No contexto da análise de sistemas há uma tarefa conhecida como sendo a *passagem* do mundo *lógico*, que trata dos requisitos conceituais, para o mundo *físico*, que trata dos requisitos e restrições tecnológicas de projeto. É conhecida também como a transição da *análise* para o *projeto.*

Há roteiros para se efetuar esta transição. Dois deles, clássicos, são conhecidos como *Análise de Transformação* e *Análise de Transação* [Jones80] [Amaral88]. Elas são aplicadas na transformação de representações gráficas de um diagrama de fluxo de dados para um diagrama de estrutura baseando-se na forma do diagrama de fluxo de dados. Sua prática, por não ter trazido resultados satisfatórios, já foi há algum tempo abandonada pelas maiorias das empresas em suas metodologias de desenvolvimento de sistemas.

Esta transformação de representação do *lógico* para o *físico* não pode ser realizada automaticamente. Não dá para ser efetuada por um processo determinístico tal como é, por exemplo, uma compilação. Um compilador faz a transformação de representação de uma

linguagem formal de alto nível para outra de baixo nível. A transformação do *lógico* para o *físico* necessita de pessoas para ser realizada.

O diagrama de estrutura é uma ferramenta gráfica interessante e adequada, tanto na abordagem sistêmica quanto na orientação a objetos, para representar os componentes de software necessários ou, os já existentes de uma aplicação de software.

A *Análise de Componentes* é a proposta de se efetuar esta transformação de representação utilizando o diagrama de estrutura.

A *Análise de Componentes* visa a determinar quais módulos de software são necessários para construir uma aplicação. Pode também ser utilizada para determinar quais módulos de software uma aplicação existente possui. A primeira, é aplicada em circunstâncias de processo de desenvolvimento de software. A segunda, em processos de *Engenharia Reversa* [Pressman97] , de *Arqueologia de Sistemas* [McMenamin84] ou de *Paleontologia Funcional* [Antón01], para estudar, entender e documentar uma aplicação de software existente.

Análise de Componentes: a partir de uma aplicação de software já existente

A etapa inicial da *Análise de Componentes* efetuada a partir de uma aplicação de software já existente, consiste em um conjunto de 6 passos para determinar as *Informações Oferecidas (IO), Informações Armazenadas (IA)* e as *Informações Processadas (IP).* Esses passos são descritos a seguir:

Passo 1. Levantar todas as saídas existentes tais como relatórios impressos, planilhas impressas, telas ou janelas de entradas ou de consultas de dados ou qualquer outro espaço lógico de interação. Selecionar uma amostra real de cada tipo.

Passo 2. Para cada saída, desdobrar o conjunto de informações que estão empacotadas nessa saída, em relações de dados semanticamente semelhantes, ou seja, em

elementos ou estruturas de dados [Warnier84][Warnier85]**.** Considerar cada informação assim descoberta em uma *IO - Informação Oferecida*.

Passo 3. Levantar todos os arquivos de dados existentes e todas as bases de dados. Para cada um deles selecionar suas definições, se existirem e tiverem sido declaradas em uma linguagem de definição de dados (*LDD*) qualquer. Para cada definição desdobrar o conjunto de informações em elementos ou estrutura de dados. Cada estrutura será uma *IA - Informação Armazenada*.

Passo 4. Para cada programa, levantar o objetivo e as funcionalidades que estão incorporadas a ele. Determinar o grau de coesão [Jones80] de cada um deles.

Passo 5. Para os programas com coesão funcional, seqüencial, comunicacional e procedural, cada funcionalidade será uma *IP - Informação Processada*.

Passo 6 Para os demais programas, que possuem coesão lógica, temporal ou coincidental haverá a necessidade de uma investigação mais detalhada no código fonte. Esta quantidade deve representar 20% do todo, pois vale a regra de Pareto [Scholtes88], conhecida como regra dos 80/20 (distribuição normal). Esta tarefa, de leitura de código fonte, fica facilitada pois, aproximadamente 80%, da aplicação já deve ter sido compreendida.

As demais etapas da *Análise de Componentes* são as mesmas tanto para o desenvolvimento de uma nova aplicação como para aplicações de software já existentes. Essas etapas consistem em projetar *DFDs* a partir das *IOs* e, projetar *DESs* a partir dos *DFDs*. Segue-se o detalhamento dessas etapas.

Análise de Componentes - etapas

Etapa 1. Para cada *IO*, *Informação Oferecida*, desenha-se um *DFD*, com uma única função, utilizando-se uma estratégia *bottom-up*, da seguinte maneira:

1.1. A principal saída, a *IO*, que só pode ser uma ou mais estrutura de dados e/ou um ou mais elementos de dados, será um fluxo de dados. Nomeado com um substantivo representando a informação, com origem na função e destino em uma *Entidade Externa*, de referência *Z*, esta nomeada também com um substantivo, representando o usuário ou cliente da informação. Na Figura 3.6, isso está representado, genericamente como o fluxo *c* e entidade *Z*. Este cliente ou usuário pode ser uma pessoa, outro processo, outro sistema, outro objeto, um equipamento ou outra coisa qualquer que assuma o papel de destinatário, de receptor da informação.

1.2. As demais saídas, consideradas saídas secundárias, serão também fluxos de dados, nomeados com substantivos representando a informação, com origem na função e destino em outro processo ou em um depósito de dados. Na Figura 3.6 estão representados como os fluxos *d* e *f*. Notar que é possível uma *IO* ter mais de um destino. Por exemplo, a informação *d* é oferecida a uma entidade *Z* e é também armazenada em um depósito *D*1.

1.3. Cada entrada necessária para geração das saídas será um fluxo de dados. Sua origem pode ser uma entidade externa, outro processo ou um depósito de dados. Os fluxos têm como destino a própria função. Eles estão representados na Figura 3.6 como os fluxos *a*, *e* e *b*.

1.4. Nomear a função. Se a *IO* for disponibilizada através de *IP*, o nome da função pode ser o nome do processamento verbalizado mais o nome da *IO*. O verbo deverá estar na forma ativa, por exemplo *calcular*, ou na terceira pessoa do singular do presente do indicativo, por exemplo *calcula*. Se IO for igual a IA pode ser utilizado o verbo *oferecer* mais o nome da *IO*, um substantivo. No *DFD* genérico da Figura 3.6 pode ser *Calcular c, Determinar c ou Oferecer c.*

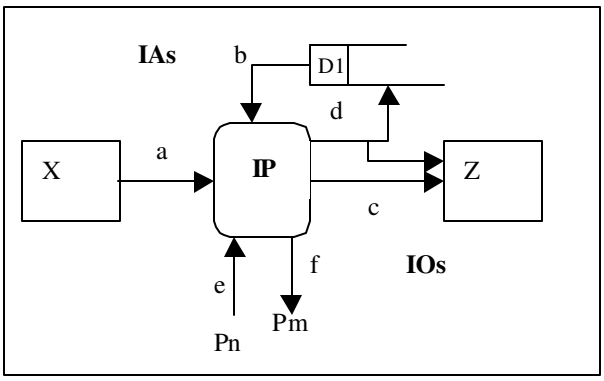


Figura 3.6 - DFD por IO genérico

Etapa 2. Para cada *DFD* obtido na etapa 1, projetar um *Diagrama de Estrutura*, o *DES*.

Para se desenhar gradualmente o *DES*, utilizando-se ora uma estratégia *bottom-up,* ora *top-down*, determinam-se os componentes de software necessários.

Propõe-se categorizar os componentes de software em quatro tipos:

- Componentes *Essenciais*,
- Componentes para *Transporte,*
- Componentes para *Controle de Padrão* e
- Componentes *Administrativos*.

Os componentes *Essenciais* são relativos ao negócio em si e os demais são relativos à tecnologia de forma geral. Em termos de projeto de componentes, mas refletido no código formal, nota-se em média, que 30% dos componentes são relativos a regras de negócio e 70% são relativos a infra-estrutura tecnológica [Penna00].

A tecnologia de informação está baseada no computador. Mesmo sendo ele um sistema complexo [Simon69] pode ser resumido em suas funções essenciais de processamento, armazenamento, controle e transferência de dados [Stallings96]. Assim, por mais que tenha sofrido, aparentemente, uma constante mudança e apresente-se hoje no mercado, em uma alta multiplicidade de logos e num amplo espectro de termos técnicos e

mercadológicos diferentes, a tecnologia suporta as mesmas tarefas básicas de infra-estrutura. São elas:

Transporte, que consiste em transferir o dado de um lugar a outro. São por exemplos os *reads*, *writes, finds, gets*, *puts*, e outros que aparecem espalhados nos programas de computadores. Sistemas de redes de comunicação também suportam atividades de transporte.

Controle de Padrão, que consiste em criticar, consistir, analisar sintaticamente, editar ou formatar um dado. São, por exemplo, as checagens de numéricos, alfabéticos, teste de dígitos verificadores, controle de totais, etc.

Administrativos, que consistem no conjunto de condições que, segundo seus valores levam a ações diferentes. São representados, por exemplo, pelo conjuntos de indicadores, conjunto de *ifs* tabelas de decisão, etc.

Assim, baseando-se nestes parâmetros, determinam-se então os componentes de software da aplicação dos quatro tipos. Segue a descrição de procedimentos para cada tipo.

2.1. Determinar os ***Componentes Essenciais***. A proposta consiste em que para cada saída da função, explicitada no *DFD* e ainda, utilizando-se uma estratégia *bottom-up*, cria-se um componente com o nome *Verbo Escolhido* mais *Nome Saída*. A proposta para o *Verbo Escolhido* pode ser conforme a lista:

- Oferecer. Quando *IO* for igual a *IA*. Exemplo: *IO = Nome Estudante* , *IA = Nome Estudante*, IP = nenhuma.

- Caracterizar. Quando a saída é qualificada pela função assumindo valores diferentes e pré-estabelecidos constantes em relação biunívoca. Exemplo: entra, em um hotel, por exemplo, *Solicitação do Cliente*, e sai *Serviço Solicitado*.

- Aprovar. Quando a saída é apenas qualificada segundo um critério pré-estabelecido. Exemplo: entra *Solicitação de Crédito*, em um agente financeiro, por exemplo, e sai *Crédito Aprovado*.

- Determinar. Quando a saída é obtida através de critérios analíticos de comparação. Exemplos: *maior salário da empresa*; *professores disponíveis em um determinado horário*.

- Calcular. Quando a saída é obtida através de um ou mais cálculos matemáticos. Exemplo: *média dos salários da empresa*.

2.2. Determinar Componentes de ***Controle de Padrão***.

2.2.1. Para a *IO* da função pode se criar, conforme descrição a seguir, ainda se utilizando uma estratégia *bottom-up*, um ou mais componentes com *Verbo Escolhido* mais nome da *IO*. A proposta para o *Verbo Escolhido* pode ser:

- *Editar, Traduzir ou Transformar*. Quando a tecnologia em que a entidade *Z* é implementada, destino da *IO*, for diferente daquela em que a função é implementada e, há a necessidade de uma interface para viabilizar a interação entre as duas. Exemplo: a função é implementada na *L4G Natural* e a entidade *Z* é um *DBMS-ORACLE*.

- *Formatar*. Quando o formato, posições de campos, códigos, valores e outros, esperados pelo cliente, o processo em *Z*, for diferente daquela que a função gerou. Exemplo: *IO* é uma estrutura de dados contendo, na geração da função os elementos na ordem *Nome País, Sigla de UF, Nome Município, Número CEP* e a entidade *Z* espera receber na ordem *Número CEP, Nome Município, Sigla da UF, Nome País*.

As informações de saída com destino a uma outra função não geram componente de software do tipo *Controle de Padrão*, pois existirá um componente, do tipo *transporte* e/ou *administrativo*, quando essa informação for tratada como entrada em uma outra função.

2.2.2. Para cada fluxo de dados de entrada com origem em uma entidade externa, utilizando-se ainda de uma estratégia *bottom-up*, cria-se também um ou mais componentes de software do tipo Controle de Padrão, nomeado de *Verbo Escolhido* mais o nome da entrada.

A proposta para o *Verbo Escolhido* pode ser:

- *Validar*. Quando a entrada real deve ser checada por meio de um padrão genérico esperado. Exemplo: o dado de entrada deve ser numérico, alfabético, deve ser checado um ou mais dígitos verificadores, ou obedecer a um critério de validação simples, como faixa de valores, limites inferiores e/ou superiores, máximos e/ou mínimos ou outros.

- *Verificar*. Quando a entrada real deve ser checada através de comparação com outro valor obtido em um arquivo, banco de dados ou tabela, para verificar existência, integridade de conteúdo ou coerência com outras informações. Essa checagem é chamada de *crítica cruzada*. Exemplo: Se *Código de Aluno* for inferior a *3000* ele deve estar matriculado na disciplina *Biologia IV*.

Os componentes de software criados a partir do descrito, tendo como verbo *validar* ou *verificar*, têm como saída um ou mais indicadores representando o diagnóstico da validação ou verificação.

2.3. Determinar os ***Componentes de Transporte***. Esses componentes de software representam os diversos métodos de acesso aos dados necessários para o funcionamento da função. Para cada fluxo de dados da função, pode-se criar, conforme descrição a seguir, utilizando-se ainda de uma estratégia *bottom-up*, um ou mais componentes com *Verbo Escolhido* mais nome da *IO*.

2.3.1. A proposta para o *Verbo Escolhido* a partir das saídas podem ser:

- *Exibir*. Quando o destino da informação, entidade *Z*, for implementada por meio de uma ou mais instâncias da classe de objetos tipo janelas ou telas, espaços lógicos de interação, de saída.

- *Imprimir*. Quando o destino da informação, entidade *Z*, for implementada por meio de uma ou mais instâncias da classe de objetos tipo relatório ou formulário impressos por um dispositivo de saída.

- *Armazenar*. Além da *IO*, este verbo é utilizado para nomear um componente de software para cada informação de saída com destino a um depósito de dados.

- *Transferir ou Transportar*. Quando o fluxo de dados de saída for implementado por meio de rede de comunicação.

Os componentes de software criados a partir do descrito, tendo como verbo *exibir, imprimir, armazenar, transferir ou transportar*, não possuem saídas informacionais. Podem ter, e isto é válido para qualquer tipo de componente de software, um ou mais indicadores (*return-code*) representando o sucesso ou não da operação para efeito de retorno de execução.

2.3.2. A proposta para o *Verbo Escolhido* a partir das entradas pode ser:

- *Exibir*. Quando, para se obter os dados de entrada, utilizam-se de uma ou mais instâncias da classe de objetos tipo janelas ou telas, espaços lógicos de interação, com campos vazios, tipo *data-entry*, que serão preenchidos em seus conteúdos através de processo de digitação.

- *Transferir ou Transportar*. Quando o fluxo de dados de entrada for implementado através de uma rede de comunicação.

- *Capturar*. Quando o fluxo de dados possuir como origem uma entidade externa.

- *Obter*. Quando o fluxo de dados possuir como origem um depósito de dados ou outra função.

Os componentes de software criados a partir do descrito, tendo como verbo *exibir, capturar, obter, transferir ou transportar*, não possuem normalmente entradas, possuindo apenas saídas representando os dados obtidos, capturados ou transferidos.
No entanto, em alguns casos, pode-se ter como entrada, um ou mais argumentos de pesquisa, normalmente códigos, representando uma ou mais chaves de acesso a um arquivo ou banco de dados qualquer. Exemplo: através de *Código Aluno*, entrada do componente de software, obtêm-se, através de acesso a um banco de dados, o *Nome do Aluno* e o *Endereço Aluno*, saídas do componente de software em questão.

2.4. Determinar os ***Componentes Administrativos***. Os componentes administrativos são utilizados para agrupar os diversos componentes, já determinados e especificados segundo um nome, sua(s) entrada(s) e sua(s) saída(s), em um *Diagrama de Estrutura*, representando as associações e comunicações entre eles.

Utilizando-se ainda de uma estratégia *bottom-up*, para cada conjunto de componentes que contribuam semanticamente para o mesmo objetivo, deve-se agregar um componente administrativo utilizando como nome um *Verbo Escolhido* mais o nome da informação obtida através do desempenho do conjunto de componentes. Assim, por exemplo, tendo-se: *Capturar a, Obter a e Validar a* agregam-se esses três componentes sob um componente administrativo, hierarquicamente superior de nome *Receber a*. O *Verbo Escolhido* podem ser:

-*Escolher ou Selecionar*. Quando os componentes subordinados são alternativos.

-*Enviar*. Quando os componentes subordinados têm características de saída. Quando os componentes subordinados possuem normalmente em seus nomes os verbos *exibir, imprimir, editar, transportar, transferir, armazenar, formatar* ou *traduzir.*

- *Receber*. Quando os componentes subordinados têm características de entrada. Quando os componentes subordinados possuem normalmente em seus nomes os verbos *exibir, transportar, transferir, validar, verificar, obter* ou *capturar*.

- *Gerar*. Quando os componentes subordinados são predominantemente do tipo essencial. Quando os componentes subordinados possuem normalmente em seus nomes os verbos *oferecer, caracterizar, aprovar, determinar* ou *calcular*.

- *Produzir ou Constituir*. Esse verbo mais o nome da informação principal, que é a *IO*, deve ser o componente administrativo de mais alto nível. Utilizando-se de uma estratégia *top-down* agregam-se a esse componente todos os outros componentes administrativos determinados e os não administrativos, que não foram agregados a nenhum outro componente administrativo.

A Tabela 3.2 abaixo resume os nomes propostos de *Verbo Escolhido*. Por *E/P/S* entende-se: características predominantes de *Entrada,* de *Processamento* e de *Saída,* respectivamente.

Tabela 3.2 - Verbos propostos para uso nos nomes dos componentes

Verbo	**Tipo**	**E/P/S**		**Verbo**	**Tipo**	**E/P/S**
Oferecer	essencial	P		Transferir	transporte	E/S
Caracterizar	essencial	P		Transportar	transporte	E/S
Aprovar	essencial	P		Imprimir	transporte	S
Determinar	essencial	P		Armazenar	transporte	S
Calcular	essencial	P		Capturar	transporte	E
Cadastrar	essencial	P		Obter	transporte	E
Editar	controle padrão	S		Constituir	administrativo	P
Traduzir	controle padrão	S		Escolher	administrativo	P
Transformar	controle padrão	S/P		Selecionar	administrativo	P
Formatar	controle padrão	S		Enviar	administrativo	S
Validar	controle padrão	E		Gerar	administrativo	P
Verificar	controle padrão	E		Produzir	administrativo	P
Exibir	transporte	E/S		Receber	administrativo	E

O roteiro apresentado não constitui um algoritmo determinístico. O produto final deste processo é um modelo representando os componentes de software, suas associações e comunicações para a implementação de uma função. Essa função pode ser uma operação de uma classe, um método de uma operação, uma atividade ou uma ação qualquer. A garantia da qualidade deste modelo é efetuada através de tarefas de revisões, validações e verificações. O modelo produzido deve ser tratado como uma *versão zero* que deverá sofrer vários ajustes de melhorias, de forma *ad-hoc*, para a geração da versão final.

A Figura 3.7 mostra um DES possível, obtido a partir do DFD genérico de *IO* da Figura 3.6.

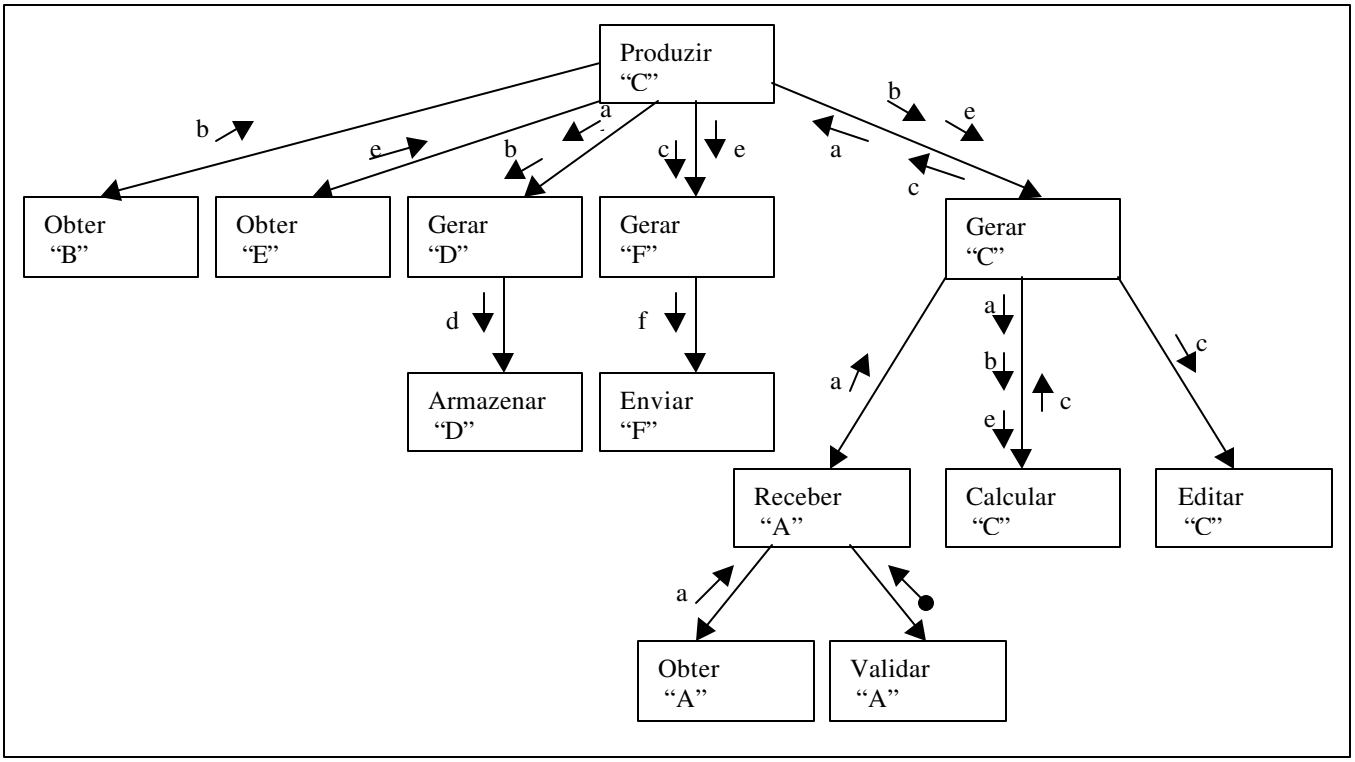


Figura 3.7 - DES genérico

3.8. Aplicação do Diagrama de Causa e Efeito

Propõe-se aplicar o Diagrama de Causa e Efeito, mostrado na Figura 3.3, no contexto de desenvolvimento de software da seguinte maneira:

. o *Resultado* pode representar *o produto de software,* mesmo não havendo ainda um consenso se este tipo de produto, software, é um bem ou um serviço. Em Yourdon [Yourdon96] afirma-se que *software está se tornando commodity.*

. a *Matéria Prima* pode representar os requisitos/restrições,

. a *Máquina* pode representar os ambientes de hardware e software,

. a *Mão de obra* representar os recursos humanos e

. o *Método* pode representar as técnicas, esquemas, roteiros e metodologia adotada.

RESULTADO: produtos de software

Para cada produto ou tipo de produto de software, pode ser estabelecida uma política de fornecimento [Sommerville01], tal como desenvolver, encomendar (terceiro) ou adquirir pronto. Se a opção for desenvolvimento próprio, este trabalho deve ser planejado. Os requisitos/restrições administrativas referente ao tempo deste desenvolvimento podem ser representados em um diagrama tipo cronograma por atividade ou por produto. Um *produto* pode ser uma unidade operacional. Uma unidade operacional é uma parte da aplicação que pode funcionar de forma independente.

Propõe-se um requisito administrativo para toda unidade operacional. Esta característica está vinculada à quantidade de ocorrências de falhas na aplicação de software. É um indicador de qualidade valorizado após a aplicação entrar em operação. É nomeado de *GA - Grau de Atenção*.

Esse *Grau de Atenção* pode ser representado por valores numéricos, por conceitos alfabéticos ou cores estabelecidas em uma escala de valores. Pode-se atribuir o valor inicial baseando-se apenas no sentimento da equipe técnica. Conforme a periodicidade e quantidade de defeitos, interrupções, reclamações ou quaisquer outras ocorrências imprevistas que forem surgindo e, removidas suas causas, o valor do *Grau de Atenção*, vai sendo baseado em critérios pré-estabelecidos, incrementado ou decrementado conforme o caso.

Divulgado de maneira padrão, por exemplo, no canto superior direito de todas janelas da aplicação, este indicador pode trazer benefícios. Para a equipe técnica serve como um indicador da qualidade. Para o corpo administrativo serve como valor de avaliação e negociação de prazos. Para a comunidade usuária serve como alerta no uso dos produtos oferecidos pela aplicação de software e de evitar falsas expectativas quanto ao desempenho dessa mesma aplicação.

A classificação de ocorrências propostas a seguir pode ajudar na atribuição dinâmica dos valores do indicador *Grau de Atenção* da aplicação: ocorrência prevista, ocorrência provocada ou ocorrência imprevista.

A ocorrência *prevista* significa que ocorre alguma quebra de regra de restrição de integridade padrão e existe, implementado no software, um mecanismo de controle deste padrão. A manifestação pode ser uma interrupção temporária no processamento do programa acompanhada ou não de mensagens de alerta específicas ou genéricas. A causa dessas interrupções podem ocorrer devido a erros de operação como, por exemplo, carga de versão errada de um banco de dados e outros.

A ocorrência *provocada* significa que a aplicação sofre uma interrupção proposital devido a alguma suspeita de erro operacional ou informacional. Uma falha informacional é devido a defeito que afeta o produto final, ou seja, a informação oferecida. Um determinado programa qualquer entrou em *loop* ou a aplicação está executando um cálculo de maneira

indevida provocando a oferta de informações incorretas, e podem ser exemplos capazes de interromper uma determinada aplicação de software.

A ocorrência *imprevista* significa que a aplicação sofreu uma interrupção disparada por execução de fim anormal de um programa ou, por mecanismos de controle automático de padrões de aplicações de software de mais baixo nível, normalmente o sistema operacional.

MATÉRIA-PRIMA: requisitos/restrições

Como neste valor de *Tipo-Plano* vale a perspectiva administrativa, os requisitos/restrições devem ser tratados sob o aspecto de volume da demanda de trabalho exigido.

As demandas de serviços de informática, lista de requisitos/restrições a serem alterados, incluídos ou excluídos, são normalmente planejadas considerando quatro estados: solicitada, em andamento, cancelada e concluída. Planejamentos assim ficam bastante distantes do realizado já que, na verdade, esses estados representam apenas as necessidades visíveis. As invisíveis representam 5.3 vezes mais do que as conhecidas [Yourdon89]. Assim, propõe-se o modelo representado na Figura 3.8 que mostra um *DTE - Diagrama de Transição de Estados*, para a entidade requisito/restrição ampliado do modelo de Yourdon para *backlog* [Yourdon89]. Pode ser utilizado como referência nas atividades de planejamento e administração das demandas.

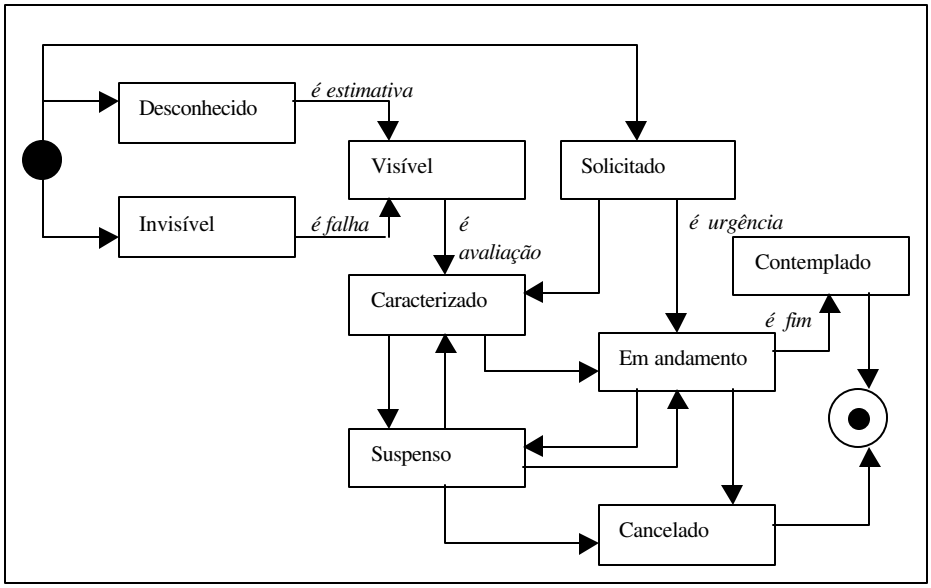


Figura 3.8 - Estados propostos dos requisitos/restrições

No estado **Desconhecido** os requisitos/restrições são na realidade ainda requisitos em potenciais. Podem ser apenas estimados baseados em aspectos externos à organização. São requisitos dependentes de governo, legislação, fornecedores, mercado, clientes finais e outros. Cenários podem ser idealizados utilizando-se de técnicas de *Planejamento Estra*tégico [Certo90] e através deles efetuarem-se algumas estimativas.

No estado **Invisível** as necessidades estão escondidas na própria aplicação de software. São os relativos à remoção de defeitos em potenciais que existem na aplicação e que ainda não se manifestaram, através das falhas, ou ainda não foram detectados. Existem também aqueles que são dependentes de outros requisitos e, não foram diretamente explicitados; aqueles que ficam subentendidos.

No estado **Visível** os requisitos/restrições representam demandas já explicitadas relativas às soluções provisórias que necessitam de serem transformadas em soluções

permanentes. Representam também a demanda necessária para remoções de defeitos já conhecidos.

No estado **Solicitado** o requisito/restrição já está explicitado. Já representa uma intenção de contemplá-lo. Muitas vezes, o requisito solicitado manifesta-se na solicitação, efetuada de maneira formal ou não, e já como uma solução e não como um problema a ser resolvido.

O estado **Caracterizado** é alcançado após uma avaliação segundo o ângulo e valores da informática. São então planejados e priorizados os serviços de informática necessários para contemplar o requisito. Na Tabela 3.3 sugerem-se algumas caracterizações conforme quatro aspectos. O terceiro aspecto é discutido em [Hatalsky02] e, o quarto aspecto *natureza da solução técnica* pode ser visto em [Dekleva92].

Tabela 3.3 - Itens propostos para caracterização de uma demanda

Aspectos			
Quanto a urgência	**Quanto a clareza**	**Quanto a necessidade**	**Natureza da solução técnica**
Oportuna	Completa	Necessária	Adaptativa
Inoportuna	Incompleta	Desejável	Corretiva
Imperativa	Confusa	Supérflua	Aperfeiçoadora

No estado **Em andamento** o requisito está sofrendo o tratamento necessário, pela equipe técnica, para ser contemplado. Já representa um trabalho a ser realizado.

No estado **Contemplado** o requisito já foi agregado à aplicação de software. O trabalho realizado para tal é considerado concluído.

No estado **Suspenso** o trabalho necessário para agregar o requisito foi interrompido por qualquer razão. Porém, existe ainda intenção de contemplá-lo oportunamente.

Neste estado **Cancelado** o requisito não será mais agregado na aplicação de software. O trabalho necessário para contemplá-lo foi definitivamente cancelado.

MÁQUINA: hardware/ software

O levantamento de requisitos administrativos relativos a este item, máquina, deve ser efetuado tendo como referência as necessidades de recursos de hardware/software.

Ferramentas de software para auxílio do desenvolvimento, tais como *CASE* [Gane88][Quatrani00] e outras podem também ser discutidas dentro deste item.

MÃO de OBRA: peopleware

Sobre os aspectos administrativos, este item diz respeito às pessoas que direta ou indiretamente representam o terceiro recurso denominado *peopleware* [DeMarco87] , *stakeholders* (*users, customers, developers,maintainers*) [Sommerville97][In01] ou *workers* [Jacobson01].

Formação de equipes, alocação de recursos, distribuição de responsabilidades e de papéis, controle de disponibilidades, levantamento de habilidades, previsão de férias, plano de treinamento, controle de desempenho, são representações de requisitos/restrições administrativas obtidos através da referência a este item.

MÉTODO: esquemas / técnicas / metodologias

Estabelecer, selecionar e adotar técnicas, métodos, metodologias, esquemas de trabalho, princípios e diretrizes são representações de requisitos/restrições administrativas referente ao método.

O processo conhecido pela sigla *RUP - Rational Unified Process* [Jacobson01] oferece uma referência de um processo genérico (*framework*), com significativa independência de técnicas ou métodos. Chega até a uma certa independência do tipo de produto, do resultado final deste processo. O *RUP* pode ser útil e aplicável também em contextos diferentes do desenvolvimento de aplicações de software. Um ponto importante no *RUP* é que um projeto específico de desenvolvimento de uma aplicação de software é uma instância deste processo. O que isto significa? Significa que para cada projeto específico há uma metodologia específica. Para cada projeto constrói-se, digamos, uma nova metodologia. Pode-se ter então, tantas metodologias diferentes quantos forem os diferentes projetos de desenvolvimento de software. Os requisitos/restrições do *Tipo-Plano* de valor *Administração*, relativos ao método, podem então ser obtidos na tarefa de configuração dos valores de atributos para, a partir do *RUP*, instanciar um processo específico para o projeto.

CAPÍTULO 4 - Sugestões de representações

4.1. Representações para Limite

A Tabela 4.1 sugere possibilidades para *Referência-Representação* e *Tipo-Representação* para o componente *Limite* conforme alguns valores de *Tipo-Foco* e *Tipo-Plano*.

Tabela 4.1 - Sugestões para representação de Limite

Referência Representação	**Nome Representação**	**Tipo Representação**	**Tipo Abordagem**	**Tipo Foco**	**Tipo Plano**
DLS	Diagrama de Limite do Sistema	gráfica	Sistêmica	Dado	Utilização
DPA	Diagrama de Pacotes	gráfica	Orientada a Objetos	Função	Utilização Produção
RELA Eventos	Relação de Eventos	textual	Sistêmica	Evento	Utilização Software Produção
RELA Classes	Relação de Classes	textual	Orientada a Objetos	Dado	Software Utilização Produção

Um *Limite* pode ser determinado e representado, pela abordagem sistêmica, através do *DLS*, *Diagrama de Limite do Sistema*, apresentado no Capítulo 2.

Ainda, pela abordagem sistêmica o *Limite* pode também ser determinado e especificado por uma *Relação de Eventos* a que o sistema, de uma forma ou outra, pretende reagir. Um limite assim determinado já está de certa forma analisado, ou seja,

particionado por eventos. No Capítulo 3 é detalhado o processo de como produzir uma *Relação de Eventos* utilizando-se da técnica proposta denominada de *Análise dos Eventos*.

Pela abordagem orientada a objetos o *Limite* pode ser definido através de uma *Relação de Classes*. Essa relação é uma lista, contendo em cada item um substantivo seguido opcionalmente por um adjetivo nomeando a classe. Daí considerar-se o *Tipo-Foco* com valor *Dado* para esta sugestão, já que as características comportamentais das classes ainda não são explicitadas. O *Limite* da aplicação fica estabelecido por este conjunto de classes.

Ainda pela abordagem orientada a objetos há um *Diagrama de Pacotes* [Fowler00] que representa grupo de classes, subsistemas ou macro funções sob um nome geral e suas dependências. A dependência entre pacotes sugerida por *Fowler* lembra em muito a idéia de acoplamento imagem [Jones80], que consiste no compartilhamento de uma estrutura de dados por diversas funções e que pode acarretar a propagação de necessidade de mudança em todos componentes, quando se altera qualquer item da estrutura compartilhada por eles. O conteúdo dos pacotes pode ser outros pacotes ou uma lista de classes. Notar que ao possibilitar que um pacote tenha como conteúdo outros pacotes utiliza-se da estratégia *top-down* de detalhamento proposta pela *Análise Estruturada* [Gane79]. Na orientação a objetos não existe a abstração de detalhamento de uma classe em outras classes detalhadas. O que chega mais perto de um detalhamento de classes é a idéia de especialização de uma classe em subclasses [Szmit93].

4.2. Outras representações sugeridas

A Tabela 4.2 sugere, nas várias representações gráficas, na abordagem sistêmica ou orientada a objetos, o *Tipo-Foco* de valores: função, dado, controle, comportamento e o *Tipo-Plano* de valores: software, utilização, produção ou administração, este último referenciado simplesmente pelas letras maiúsculas *S, U, P* ou *A*, respectivamente.

Tabela 4.2 - Sugestões de abordagem, plano e foco para várias representações gráficas

Ref.	Representação	Abordag.	Foco	Plano
DFD	Diagrama de Fluxo de Dados	Sistema	Função	U
DFC	Diagrama de Fluxo de Controle	Sistema	Controle	S, P
DCU	Diagrama de Caso de Uso	Sistema	Função	S, U
DCL	Diagrama de Classes	Objeto	Função, Dado	S, U
DAC	Diagrama de Associação de Classes	Sistema	Dado	S, U
DAT	Diagrama de Atividades	Objeto	Função, Controle, Comp.	S, U
DES	Diagrama de Estrutura	Objeto	Função	S
DSQ	Diagrama de Seqüência	Sistema	Dado, Evento, Controle, Comp.	S, U, P
DTE	Diagrama de Transição de Estados	Objeto	Comp.	S, U, P
DER	Diagrama de Entidades e Relacionamentos	Sistema	Dado	U
DDI	Diagrama de Diálogo	Sistema	Comp.	U, P
DCO	Diagrama de Colaboração	Sistema	Comp.	S, U, P
DPA	Diagrama de Pacotes	Sistema	Função	S, U
DBL	Diagrama de Blocos	Objeto	Função, Controle	S, U
DJA	Diagrama de Jackson	Objeto	Função, Controle	S, U
DNS	Diagrama de Nassi-Shneiderman	Objeto	Função	S, U
DRE	Diagrama de Reação a Eventos	Sistema	Função, Evento, Comp.	S, U
DLS	Diagrama de Limite do Sistema	Sistema	Dado	U
DCE	Diagrama de Causas e Efeitos	Sistema	Dado	S, U, P, A
CRO	Cronograma	Sistema	Função, Dado	A

Na *Análise por Visões* não é a ferramenta de representação que é categorizada dentro de uma visão (abordagem , estratégia, limite, foco e plano). O que se enquadra em uma visão são os diversos requisitos/restrições que a aplicação pretende contemplar. Porém, este enquadramento apresentado na Tabela 4.2 pode auxiliar na escolha da ferramenta de representação de um conjunto de requisitos/restrições. Esta tabela pode também ser incrementada ou alterada. Algumas observações, apresentadas a seguir, podem também ajudar a justificar certas escolhas.

O DCU - Diagrama de Caso de Uso apresentado originalmente no contexto da orientação a objetos é aqui tratado como abordagem sistêmica. Ele representa macro-funcionalidades. Mesmo sem o interesse de expressar a interação entre essas funcionalidades e de ter a finalidade de permitir a futura descoberta de objetos que a viabilizem, ele abstrai bem a idéia de um ambiente e de algo limitado que interage com este ambiente. Essas abstrações são claramente sistêmicas.

O DAC - Diagrama de Associação de Classes foi considerado pela abordagem sistêmica e *Tipo-Foco* de valor *Dado* pois ele expressa apenas o nome genérico da classe e sua(s) associação(ões) com outras, ficando assim muito semelhante ao DER – Diagrama de Entidades e Relacionamentos.

O DES - Diagrama de Estrutura é tratado como abordagem de valor orientada a objetos pois ele é mais adequado para representar pequenas funções do que macrosfunções como foi originalmente apresentado. É adequado para representar, por exemplo, métodos de uma determinada operação de uma classe ou mesmo a interação de operações intraclasses. A mesma consideração vale para os *Diagramas de Blocos*, referência DBL, de *Jackson*, referência DJA e de *Nassi*, referência DNS. Esses últimos já foram originalmente recomendados para serem utilizados em nível de programa e não para tratar macros funcionalidades ou processos.

Os valores para *Tipo-Foco* foram atribuídos pelas representações predominantes que a ferramenta expressa. Não significa que ela não possa mostrar também, secundariamente, outros valores de *Tipo-Foco*.

A maioria das ferramentas foram enquadradas para representação do *Tipo-Plano* de valor *Utilização*, requisitos/restrições conceituais, requisitos isentos de tecnologia.

A Tabela 4.3 sugere, nas várias representações textuais ou mistas, categorizadas em *Relação, Descrição, Atribuição e Especificação*, conforme proposto na Tabela 2.3 na abordagem sistêmica ou orientada a objetos, o *Tipo-Foco* de valores: função, dado, controle, comportamento e o *Tipo-Plano* de valores: software, utilização, produção ou administração, esses últimos referenciados simplesmente pelas letras maiúsculas *S, U, P* ou *A*, respectivamente.

Tabela 4.3 - Sugestões de abordagem, plano e foco para várias representações textuais

Referência	**Representação**	**Abordagem**	**Foco**	**Plano**
RELAÇÃO	Eventos	Sistema	Evento	S, U
	Classes	Objeto	Dado	S, U
	Cenários	Sistema	Função, Comportamento	S, U
DESCRIÇÃO	Entidade Externa	Objeto	Dado	S, U
	Ator	Objeto	Dado	S, U
	Interagente	Objeto	Dado	S, U
	Casos de Uso	Sistema	Função, Comportamento	S, U
	Funcionalidade	Sistema	Função	S, U
	Cenário	Sistema	Função, Comportamento	S, U
	Associação	Objeto	Função	S, U
	Relacionamento	Objeto	Função	S, U
	Ação	Objeto	Função	S, U
	Atividade	Objeto	Função	S, U
ATRIBUIÇÃO	Elemento de Dados	Objeto	Dado	S, U
	Estrutura de Dados	Objeto	Dado	S, U
	Fluxo de Dados	Objeto	Dado	U
	Depósito de Dados	Objeto	Dado	U
	Entidade de Dados	Objeto	Dado	S, U
	Atributo	Objeto	Dado	S, U
	Adjetivo Indefinido	Objeto	Função, Dado	S, U
ESPECIFICAÇÃO	Função	Objeto	Função	S, U
	Operação	Objeto	Função	S, U
	Método	Objeto	Função	S, U
	Serviço	Objeto	Função	S, U

As relações de *Eventos*, de *Classes* e de *Cenários* e as descrições de cada *Caso-de-Uso*, de *macro funcionalidades* e de *Cenários*, que consiste, esse último, em uma ou mais funcionalidades desejadas por meio de uma ou mais interações entre objetos, são categorizadas como representações pela abordagem sistêmica pelo fato de que o modelador, quando efetua esta modelagem, está abstraindo idéias de conjunto limitado de coisas, de pluralidade, de coisas inter-relacionadas, de interações e integrações entre essas coisas.

As descrições de *Relacionamento*, de *Associação*, de *Ação* e de *Atividade* são categorizadas dentro da abordagem orientada a objetos pois tratam-se de coisas pontuais, de partes, de componentes, de coisas singulares, atômicas e isoladas. Os mesmos comentários valem para as *atribuições* e as *especificações*.

CAPÍTULO 5 - Conclusões e sugestões para novos trabalhos

5.1. Conclusões

O fato de ter-se como princípio básico a certeza de que não há ferramenta de modelagem ruim ou obsoleta e direcionando o olhar para qualidades e limitações e não para seus defeitos, permite o resgate e uma melhor recomendação do uso do que elas têm de bom.

A adoção, por uma equipe de desenvolvedores, de um método de trabalho é um dos fatores que pode diferenciar um ambiente de trabalho profissional de um ambiente amador. Se uma equipe não apresenta um método de trabalho ela provavelmente está trabalhando sob improviso. Apesar do enorme consenso de que qualquer metodologia é melhor do que nenhuma, elas são no geral pouco adotadas. Uma das causas pode ser que as metodologias propostas são normalmente amarradas a técnicas, ferramentas e/ou notações de modelagem, exigindo do modelador um investimento razoável para o conhecimento dessas técnicas e para o aprendizado de sua aplicação. Outro fator é que elas mudam como muda a tecnologia. Esses fatores, entre outros, podem dificultar a sedimentação e melhoria de processos de desenvolvimento ao longo do tempo.

A *Análise por Visões* traz a vantagem de não estar vinculada a técnicas, ferramentas de representação e tecnologias específicas. Propõe apenas *Visões* em domínios de abstrações não fechados. Ela também é flexível. Nada impede de classificar e/ou especializar, conforme necessidades, mais valores para *Foco* ou *Plano,* além dos propostos.

Dada uma *Visão* de um domínio pode-se também indicar as ferramentas de representação existentes mais adequadas a ele ou discutir a criação de novas técnicas. Como o modelo da *Análise por Visões* é relativamente simples esta técnica possui um elevado grau de aplicabilidade prática.

A tentativa de enquadrar, dentro de uma visão, as ferramentas de modelagem mais conhecidas, permitiu constatar que elas são fartas no aspecto conceitual, requisitos ou restrições do tipo de plano de valor *Utilização*, e raras para os outros aspectos, necessidades dos tipos de plano de valor *Produção* e *Administração*.

A *Análise por Visões* também amplia, justamente através do conceito de *Planos*, o particionamento clássico, que se foi um dos mais úteis até hoje, ou seja os requisitos *Lógico* e *Físico*. Esta ampliação pode melhorar, no mínimo sob aspectos didáticos, utilizando seu modelo como um grande *guideline*, o ensino da *análise de requisitos*.

Traz também como vantagem a possibilidade de maior segmentação, facilitando como mostra a *Análise de Componentes* a representação de módulos de software. Traz como desvantagem a dificuldade de integração entre os diversos modelos gerados. Expõe-se assim, o risco de maior possibilidade de existência de *gaps* semânticos.

A *Análise por Visões* possibilita que a categorização dos requisitos seja efetuada no momento em que eles vão surgindo, nas etapas normalmente chamadas de *Estudos Iniciais*, *Elicitação* e *Levantamentos* de requisitos e não nas etapas posteriores de modelagens. Com isso, a futura Análise de Sistemas pode ser facilitada pois, possibilita também o paralelismo e especialização nas tarefas de modelagem.

5.2. Contribuições

A principal contribuição é a que dá título ao trabalho como um todo, a *Análise por Visões*. Através de seu modelo referência, Figura 3.1, pode-se orientar as atividades de levantamento e análise de requisitos.

Destaca-se a classificação original para a entidade *Plano*, bem diferente das classificações de domínios propostas especificamente para classes de objetos. Os planos de valores *Produção e Administração* contemplam requisitos/restrições *ad hoc*, tais como procedimentos que devem ser executados por pessoas, que podem gerar requisitos ou restrições e devem ser implementados no produto de software. É proposto também que o levantamento das necessidades operacionais, os requisitos/restrições do *plano* de valor *Produção,* seja efetuado a partir de cada dispositivo de entrada e saída da aplicação de software tais como os relatórios, as planilhas e as telas de interação.

É proposto que o levantamento dos requisitos/restrições do *plano* de valor *Software/construção* comece através da determinação das *informações armazenadas* (IA) e das *informações processadas* (IP) necessárias pela aplicação de software para a geração das *informações oferecidas* (IO) e já levantadas na determinação dos requisitos do *plano* de valor *Utilização*.

A *Análise dos Eventos* oferece um roteiro prático para o levantamento, análise e modelagem da reação do sistema aos eventos que determinada aplicação de software pretende contemplar. Destacam-se as propostas originais do roteiro de análise/síntese dos eventos, de classificação dos eventos, Tabela 3.2, das *oito perguntas para modelagem da reação*, e da representação gráfica, o *Diagrama de Reação a Eventos*.

A *Análise de Componentes* resgata uma ferramenta gráfica, o diagrama de estrutura, de determinação, documentação e avaliação dos módulos de software necessários. Essa ferramenta quando aplicada a um contexto já bem segmentado pode gerar modelos mais simples e mais claros. A *Análise de Componentes* também propõe um roteiro

para a geração da primeira versão do diagrama de estrutura. Propõe também uma lista de ações, Tabela 3.3, classificadas pela sua natureza tecnológica (o *como*), que devem ser agregadas a uma função essencial (o *o que*).

É sugerido o uso do *Diagrama de Causas e Efeitos* (4M), como referência para um processo de desenvolvimento de software, e facilitar o levantamento e implantação inicial deste processo. Também permite que os requisitos/restrições administrativos possam ser levantadas de uma maneira melhor organizada.

É proposto o modelo de *Estados de Requisitos/Restrições* apresentado na Figura 3.8. O modelo permite ajudar no refinamento de atividades administrativas de planejamento do trabalho de desenvolvimento de uma aplicação de software. São também apresentados, na Tabela 3.3, os aspectos para caracterizar, sob a ótica administrativa, uma determinada demanda de serviço de informática.

A proposta de adoção do indicador de qualidade denominado *Grau de Atenção* pode ajudar a minimizar alguma frustração de clientes e usuários causada por falsas expectativas em relação à qualidade do desempenho de uma aplicação de software.

As Tabelas 4.1, 4.2 e 4.3 que apresentam o enquadramento, em termos de *abordagem*, *foco* e *plano* das ferramentas de representações mais conhecidas, podem auxiliar na seleção e escolha da ferramenta de modelagem.

5.3. Sugestões para novos trabalhos

Em continuidade às propostas de técnicas e recomendações efetuadas no contexto de análise de requisitos para o desenvolvimento de sistemas de informações suportados por aplicações de software seguem-se as sugestões:

Determinar novas ferramentas de representação.

O modelo referência apresentado como *Análise por Visões* abre um leque de possibilidades de pesquisa de novas técnicas e ferramentas de representação para a modelagem de requisitos/restrições conforme os valores de *Foco* e de *Plano*. No modelo referência proposto, com seis *Planos* e cinco *Focos*, tem-se a possibilidade de trinta tipos de *visões* diferentes. Como para cada *visão* pode-se dar duas abordagens, sistêmica e orientada a objeto, em tese têm-se sessenta tipos de *visões* diferentes, cada uma definindo uma abstração.

Pode-se efetuar um estudo para verificação de abrangência, de adequação e de utilidade se esta segmentação, por *visões*, puder determinar as classes para registro e divulgação de objetos, nos moldes da *Análise de Domínios*, contribuindo para a possível reutilização de objetos de software.

Elaborar padrões de modelos de análise de requisitos.

As *visões* podem ser úteis como referência de documentação e organização de problemas encontrados no trabalho de análise de requisitos. Isto pode ser um ponto de partida para o estudo, análise e o possível estabelecimento de um conjunto de *padrões práticos de análise de requisitos*, nos moldes já existentes nos *padrões de projeto.*

Estabelecer linhas padrões de produção da informação.

Particularmente, a *Análise por Visões* abre um interesse para a pesquisa de novos modelos para requisitos do plano *Produção* e do plano *Administração*. Há também o interesse na pesquisa que resulte em metodologias para elaboração do estabelecimento e manutenção de *ambientes de produção* e de *linhas de produção* (seqüência de processos de software necessários para gerar um tipo de informação) no amplo espectro que vai da computação pessoal até a computação corporativa.

Na *visão* de plano *software/construção*, especificamente no contexto da técnica proposta como *Análise de Componentes* abre a possibilidade de ampliar e de possibilitar que seja limitado os tipos módulos de software que representam a tecnologia adotada. Talvez, um conjunto finito de módulos, que representem solução tecnológica, permitam a implementação, em fase de projeto, de qualquer módulo conceitual.

Estabelecer critérios e modelos para comparações de técnicas de análise de requisitos.

Como foi apresentado no Capítulo 2, podem ser determinados itens e critérios para possibilitar comparações entre as diversas técnicas de análise de requisitos. Sugere-se que a abstração de uma técnica/método seja representada através de um modelo de dados ou de um modelo de classes para melhorar e/ou facilitar comparações com outras técnicas/métodos modeladas da mesma maneira.

Verificar a possibilidade de utilização em outros tipos de aplicações.

A *Análise por Visões* foi elaborada para ser útil no contexto de desenvolvimento de aplicações do tipo de *sistemas de informação*. A verificação de utilidade para o desenvolvimento de outros tipos de aplicações de software necessita ser verificada.

Estabelecer critérios para verificação/avaliação da qualidade de modelos.

A *Análise por Visões* tem como princípios a coerência entre abstração e representação e que quanto menor o escopo mais fácil fica a tarefa de modelagem. Assim, dando realce à abstração, pois *visão* é um destaque da abstração, e aumentando a segmentação acredita-se na possibilidade de se obter uma melhor qualidade nos modelos gerados. O desenvolvimento de critérios de avaliação e de verificação da qualidade de modelos são necessários para atestar esses princípios.

Avaliar o *gap* semântico.

Como a *Análise por Visões* aumenta o grau de segmentação e tem como princípio a possibilidade de utilização de qualquer ferramenta de representação, tem-se como conseqüência a geração de vários modelos de tipos diferentes. Isto pode dificultar o trabalho de rastreabilidade, remoção de redundância, checagem de integridade e avaliação das conseqüências geradas pelo problema do *gap* semântico.

Ajustar a *Análise por Visões, dos Eventos e Análise de componentes* para a *UML*.

Através dos mecanismos de estereótipos e outros há a necessidade de um trabalho de avaliação e ajustes para possibilitar a inclusão dessas três técnicas na UML para facilitar sua divulgação e utilização.

Desenvolver uma ferramenta CASE.

Há a necessidade de desenvolvimento de uma ferramenta de software de apoio para a utilização da *Análise por Visões, Análise dos Eventos e Análise de Componentes*.

Experimentar a *Análise por Visões*.

A oportunidade de experimentação prática da *Análise por Visões* em projetos reais de porte significativo, de desenvolvimento de aplicações de software, é esperada e necessária para uma melhor avaliação e ajustes da mesma. Também se espera a aplicação prática do resultado da *análise de requisitos*, obtido através da *Análise por Visões*, na fase seguinte de implementação. As opiniões e sugestões de projetistas e programadores sobre os modelos gerados é de fundamental importância para uma avaliação da técnica.

REFERÊNCIAS BIBLIOGRÁFICAS

[Almendros02] Almendros- Jimenez, J.M., Luis Gonzalez- Jimenez, "Bases for the development of LAST: a formal method for business software requirements specifications", Information and Software Technology, vol.44, pp.65-75, Elsevier Science Ltd., 2002.

[Alvarez02] Alvarez, Rosio, "Confessions of an information worker: a critical analysis of information requirements discourse", Information and Organization, vol.12, pp.85-107, Elsevier Science Ltd., 2002.

[Amaral88] Amaral Filho, Antônio Rubens Anciães, "Projeto Estruturado, Fundamentos e Técnicas", Livros Técnicos e Científicos, RJ, 1988.

[Antón01] Antón, A. L. and Colin Potts, "Functional Paleontology: System Evolution as User Sees It", IEEE, Proceedings of the 23rd International Conference on Software Engineering, ICSE'01, pp. 421-432, Toronto, Canada, may, 2001.

[Argila98] Argila, Carl and Edward Yourdon, "Case Studies in Object-Oriented Analysis and Design", Prentice-Hall, 1998.

[Arango89] Arango, G. and R. Prieto-Diaz, "Domain Analysis: Acquisition of Reusable Information for Software Construction", IEEE Computer Society Press,1989.

[Ashby70] Ashby, W. Ross, "Introdução à Cibernética", Perspectiva, RJ, 1970.

[August91] August, Judy H., "JAD - Joint Application Design", Yourdon Press, NY, 1991.

[Avison03] Avison, D. E., and Guy Fitzgerald, "Where Now for Development Methodologies ?", Communications of the ACM, vol. 46, no. 1, pp.78-82, January, 2003.

[Barfield93] Barfield, Lon, "The User Interface: Concepts & Design", Addison-Wesley, 1993.

[Beck99] Beck**,** Kent, "Embrancing Change with Extreme Programming", IEEE Computer, vol.32, no.10, pp.70-77, october 1999.

[Bennaton84] Bennaton, Jocelyn, "O que é Cibernética", Coleção Primeiros Passos, Brasiliense, SP, 1984.

[Bertalanffy75] Bertalanffy, Ludwig Von, "Teoria Geral dos Sistemas",Vozes, RJ, 1975.

[Bochenski94] Bochenski, Barbara, "Implementing Production-Quality Client/Server", John Wiley & Sons,1994.

[Boehm01] Boehm, Barry and Daniel Port, "Educating Software Engineering Students to Manage Risk", IEEE, Proceedings of the 23rd International Conference on Software Engineering, pp.591-601, Toronto, Canada, may, 2001.

[Booch94] Booch, Grady, "Object-Oriented Analysis and Design with Applications", Reading, Mass., Addison-Wesley, 1994.

[Booch99] Booch, Grady, James Rumbaugh and Ivar Jacobson, "The Unified Modeling Language User Guide", Reading, Mass., Addison-Wesley, 1999.

[Bosch01] Bosch Jan, "Software Product Lines: Organizational Alternatives", IEEE, Proceedings of the 23rd International Conference on Software Engineering, pp. 91-100, ICSE'01, Toronto, Canada, may, 2001.

[Brown97] Brown, David, "An Introduction to Object-Oriented Analysis", John Wiley & Sons, 1997.

[Brown00] Brown, Steve, "Visual Basic 6.0 – Biblia do Programador", Berkeley,2000.

[Carvalho88] Carvalho, Luiz Carlos de Sá, "Análise de Sistemas - O Outro Lado da Informática", Livros Técnicos e Científicos, RJ, 1988.

[Carvalho01] Carvalho, Ariadne M. B., Thelma C. dos Santos Chiossi, "Introdução à Engenharia de Software", Editora da Unicamp, SP, 2001.

[Certo90] Certo, S. C. and J. P. Peter, "Strategic Management: A Focus on Process", McGraw-Hill, 1990.

[Champeaux93] Champeaux, Denis de, "Object Oriented System Development", Addison-Welsey, 1993.

[Champeaux02] Champeaux Denis de, "Software Engineering Considered Harmful", Communications of the ACM, vol. 45, no.11, pp.102-104, november, 2002.

[Chen77] Chen, P.P., "The Entity-Relationship Approach to Logical Data Base Design", QED Information Sciences, Data Base Monograph Series Number 6, 1977.

[Chiavenato83] Chiavenato, Idalberto, "Introdução à Teoria Geral da Administração", Makron Books do Brasil, SP, 1983.

[Chin95] Chin, Kenneth F., "A JAD Experience", Proceedings of the ACM SIGCPR Conference on Suporting teams, groups, and learning inside and outside the IS function reinventing IS, Nashiville, Tennessee, april, 1995.

[Coad91] Coad, Peter and Edward Yourdon, "Object-Oriented Analysis", Englewood Cliffs, NJ, Prentice-Hall, 1991.

[Coleman94] Coleman, Derek, Patrick Arnold, Stephanie Bodoff, Chris Dollin, Helena Gilchrist, Fiona Hayes and Paul Jeremaes, "Object-Oriented Development: The Fusion Method", Englewood Cliffs, NJ, Prentice-Hall, 1994.

[Cox86] Cox, Brad J., "Object-Oriented Programming - An Evolutionary Approach", Reading, Mass., Addison-Wesley, 1986.

[Csilag85] Csillag, J. M., "Análise do Valor / Metodologia do Valor", Atlas, SP, 1985.

[Date81] Date, C. J., "An Introduction to Database Systems", Addison-Wesley, 1981.

[Davis83] Davis, William S., "Systems Analysis and Design", Addison-Wesley, 1983.

[Davis93] Davis, A.M., "Software requirements: objects, function and states", Englewood Cliffs, NJ, Prentice Hall, 1993.

[Dekleva92] Dekleva, S. M., "The influence of the Information Systems Development Approach on Maintenance", MIS Quarterly (16:3), pp.355-372, september, 1992.

[DeMarco78] DeMarco,Tom, "Structural Analysis and System Specification", Yourdon Press, NY, 1978.

[DeMarco82] DeMarco, Tom, "Controlling Software Projects", Englewood Cliffs, NJ, Prentice-Hall, 1982.

[DeMarco87] DeMarco, Tom and Timothy Lister, "Peopleware - Productive Projects and Teams", McGraw-Hill, 1987.

[Dijkstra68] Dijkstra, E. W., "Goto Statement Considered Harmful", Communications of the ACM, vol.11, no.3, pp. 147-149, 1968.

[Dusire02] Dusire, S., M. Flynn and N. Dardenne, "Requirements Engineering - Applying Theory to Reality", Proceedings of the IEEE Joint International Conference on Requirements Engineering RE'02, p.300, Essen, Germany, september, 2002.

[Easterbrook01] Easterbrook, Steve and Marsha Chechik, "A Framework for Multi-Valued Reasoning over Inconsistent Viewpoints", IEEE Proceedings on the 23rd International Conference on Software Engineering, pp. 411-420, Toronto, Ontario, Canada, july, 2001.

[Elrad01] Elrad, Tzila, Robert E. Filman and Atef Bader, "Aspect- Oriented Programming", Comunications of the ACM, vol. 44, no.10, pp. 29-32, october, 2001.

[Ericsson84] Ericsson, K. A. and H. A. Simon, "Protocol Analysis: Verbal Reports as Data", MIT, 1984.

[Epstein73] Epstein, Isaac (org.), A. M. Turing, Abraham Moles, W. Ross Ashby, Magoroh Maruyama, Gordon Pask, Fazlollah M. Reza, Haroldo de Campos, e Jean Hyppolite, "Cibernética e Comunicação", Cultrix, USP, 1973.

[Eva01] Eva, M., "Requirements acquisition for rapid applications development", Information & Management, Breukelen, vol.39, pp.101-107, 2001.

[Faulk97] Faulk, S. R., "Software Requirements: A Tutorial Software Requirements Engineering", 2nd Ed. IEEE CS Press, 1997, pp. 128-149, 1997.

[Feitosa91] Feitosa, Vera Cristina, "Redação de Textos Científicos", Papirus, SP,1991.

[Feliciano88] Feliciano Neto, A, J. D. Furlan e W. Higa, "Engenharia da Informação", McGraw-Hill, SP,1988.

[Feliciano96] Feliciano Neto, A. e T. Shimizu, "Sistemas Flexiveis de Informações", Makron Books do Brasil, SP,1996.

[Fiorini98] Fiorini, Soeli T., Arndt von Staa e Renan Martins Baptista, "Engenharia de Software com CMM", Brasport, RJ,1998.

[Firesmith93] Firesmith, D. G., "Object-Oriented Requirements Analysis and Logical Design", NY, John Wiley & Sons, 1993.

[Fowler00] Fowler, Martin and Kendal Scott, "UML Distilled - A Brief Guide to the Standard Object Modeling Language", Reading, Mass., Addison-Wesley Longman, 2000.

[Franquemont02] Franquemont, Sharon, "Você já sabe o que fazer", Nova Era, 2002.

[Furlan94] Furlan, J. D., "Sistema de Informação Executiva", Makron Books do Brasil, SP, 1994.

[Furlan97] Furlan, J. D., "Modelagem de Negócio", Makron Books do Brasil, SP,1997.

[Gamma95] Gamma, Erich, Richard Helm, Ralph Johnson and John Vlissides, "Design patterns - elements of reusable object-oriented software", Addison-Wesley, 1995.

[Gane79] Gane, Chris and Trish Sarson, "Structured Systems Analysis: Tools and Techniques", Prentice-Hall, 1979.

[Gane88] Gane, Chris, "Computer-Aided Software Engineering", Rapid System Development Inc.,1988.

[Gane88b] Gane, Chris, "Rapid System Development", Rapid System Development Inc.,1988.

[Gause89] Gause, Donald C. and Gerald M. Weinberg, "Exploring Requirements", Donald C. Gause & Gerald M. Weinberg, 1989.

[Gause90] Gause, Donald C. and Gerald M. Weinberg, "Are Your Ligths On?", Dorset House, 1990.

[Giesen02] Giesen, J. and A. Volker, "Requirements Interdependencies and Stakeholders Preferences", Proceedings of the IEEE Joint International Conference on Requirements Engineering, RE'02, pp. 206-209, Essen, Germany, september, 2002.

[Gillenson84] Gillenson, Mark L. and Robert Goldberg, "Strategic Planning, Systems Analysis and Database Design", John Wiley & Sons, 1984.

[Goldratt90] Goldratt, Eliyahu M. and Jeff Cox, "A Meta", IMAM, SP, 1990.

[Goleman96] Goleman, Daniel, "Inteligência Emocional", Objetiva, 1996.

[Greenspan01] Greenspan J. S., "Extreme RE: What if there is no time for Requirements Engineering ? ", IEEE Proceedings of the Fifth International Symposium on Requirements Engineering, RE'01, pp. 282-283, Toronto, Canada, august, 2001.

[Guyton89] Guyton, Arthur C., "Fisiologia Humano e Mecanismos das Doenças", Guanabara, RJ, 1989.

[Haberkorn99] Haberkorn, E., "Teoria do ERP", Makron Books do Brasil, SP, 1999.

[Harel02] Harel, D. and Orna Kupfermanm, "On Object Systems and Behavioral Inheritance", Transactions on Software Engineering, IEEE, vol. 28, no. 9, pp. 889-903, september 2002.

[Harrold91] Harrold, M. J. and M. L. Soffa, "Selecting and Using Data for Integration Testing", IEEE Software, vol.8, no. 2, march 1991.

[Hatalsky02] Hatalsky Julie and Paul S. Corwin, "The Modification Process: A Practical Means to Understand and Enhance the Software Requirements Engineering Process", Proceedings of the Sixth IASTED International Conference on Software Engineering and Applications, pp. 94-99, Cambridge, Mass., november, 2002.

[Hatley87] Hatley, D. J. and I. A., Pirbhai, "Strategies for Real- Time System Specification", Dorset House, 1987.

[Hayakawa78] Hayakawa, S. I., "Language in Thought and Action", Harcourt Brace Jovanovich, New York, 1978.

[Heales00] Heales, Jon, "Factors Affecting Information System Volatility", ACM, Proceedings of the 21st International Conference on Information Systems, ICIS'00, pp.70-83, 2000.

[Hirsch02] Hirsch, Michael, "Making RUP Agile", Object Oriented Programming Systems Language and Applications - OOPSLA - Practitioners Reports, pp. 01-08, Seatle, Washington, november, 2002.

[In01] In Hob, Barry Boehm, Thomas Rodgers and Michael Deusth, "Applying WinWin to Quality Requirements: A Case Study", IEEE, Proceedings of the 23rd International Conference on Software Engineering, ICSE'01, pp.555-564, Toronto, Canada, may, 2001.

[Jackson81] Jackson, M., "Construtive methods of program design - Software Design Strategies", IEEE Computer Society, 1981.

[Jacobson92] Jacobson, Ivar , P. Jonsson and G. Overgaard, "Object-Oriented Software Engineering", Wokingham, England, Addison-Wesley, 1992.

[Jacobson01] Jacobson, Ivar, Grady Booch and James Rumbaugh, "The Unified Software Development Process", Addison-Wesley, 2001.

[John02] John, Isabel, Peter Sody and E. Tolzmann, "Efficient and Systematic Software Evolution Through Domain Analysis", Proceedings of the IEEE Joint International Conference on Requirements Engineering RE'02, pp. 237-244, Essen, Germany, september, 2002.

[Jones80] Jones, Meilir Page, "The practical guide to structured systems design", Yourdon Inc., NY, 1980.

[Jones85] Jones, Meilir Page, "Practical Project Management", Dorset House, 1985.

[Jones86] Jones, Capers, "Programming Productivity", McGraw-Hill, 1986.

[Jones95] Jones, Meilir Page, “What Every Programmer Should Know About Object-Oriented Design”, Dorset House, 1995.

[Jones00] Jones, Meilir Page, “Fundamentals of Object-Oriented Design in UML”, Addison-Wesley, 2000.

[Kettinger97] Kettinger, W., J. Teng, and S. Gusha, “Business process: a study of methodologies, techniques, and tools”, MIS Quarterly, Minneapolis, vol.21, no.1, pp. 55-80, 1997.

[Kiczales01] Kiczales, Gregor, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm and W. G. Griswold, “Getting Started with Aspectj”, Comunications of the ACM, vol. 44, no.10, pp. 29-32, october, 2001.

[Kiedaisch01] Kiedaisch, F., Martin Pohl, Joachim Weisbrod, Siegfried Bauer and Stefan Ortmann, “Requirements Archaeology: From Unstructured Information to High Quality Specifications”, IEEE, Proceedings of the Fifth International Symposium on Requirements Engineering, RE’01, pp. 304-305, Toronto, Canada, august, 2001.

[Knethen02] Knethen, A. V., Barbara Paech, Friedemann Kiedaisch and Frank Houdek, “Systematic Requirements Recycling through Abstraction and Traceability”, Proceedings of the IEEE Joint International Conference on Requirements Engineering, RE’02, pp. 273-281, Essen, Germany, september, 2002.

[Korel90] Korel, B., “Automated Software Test Data Generation”, IEEE Transaction On Software Engineering, vol. 16, no. 8, august, 1990.

[Korth86] Korth, Henry F. and Abrahan Silberschatz, “Database System Concepts”, McGraw-Hill, 1986.

[Kotonya98] Kotonya G. and Sommerville L., “Requirements Engineering: Process and Techniques”, John Wiley & Sons, 1998.

[Kovitz02] Kovitz Bem, “Ambiguity and What to Do About It”, Proceedings of the IEEE Joint International Conference on Requirements Engineering, RE’02, p. 213, Essen, Germany, september, 2002.

[Kozlenkov02] Kozlenkov, A. and Andrea Zisman, “Are their Specifications Consistent with our Requirements?”, Proceedings of the IEEE Joint International Conference on Requirements Engineering, RE’02, pp. 145-154, Essen, Germany, september, 2002.

[Kumar02] Kumar, Ram, “Managing risk in IT projects: an options perspective”, Information & Management, Breukelen, vol. 40, pp. 63-74, 2002.

[Lamsweerde00] Lamsweerde, A. V., "Requirements Engineering in the Year 00: A Research Perspective", Invited Keynote Paper, ACM, ICSE'00, Proceedings of 22nd International Conference on Software Engineering, pp. 5-19, 2000.

[Lamsweerde01] Lamsweerde, A. V., "Goal-Oriented Requirements Engineering: A Guided Tour", IEEE, Proceedings of the Fifth International Symposium on Requirements Engineering, RE'01, pp. 249-262, Toronto, Canada, august, 2001.

[Lamsweerde02] Lamsweerde, A. V. and Letier, E., "Agent-Based Tactics for Goal-Oriented Requirements Elaboration", ACM, Proceedings of 24th International Conference on Software Engineering, ICSE'02, pp. 83-93, Orlando, Flórida, USA, may 2002.

[Larman99] Larman, Craig, "Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design", Englewood Cliffs, NJ, Prentice-Hall, 1999.

[Lee02] Lee, Seok Won, "Proxy Viewpoints Model- based Requirements Engineering", Proceedings of the ACM Symposium on Applied Computing, Madri, Span, pp.1004-1008, march, 2002.

[Letier00] Letier E. and A. V. Lamsweerde, "Handling Obstacles in Goal-Oriented Requirements Engineering", IEEE Transactions on Software Engineering, Special Issue on Exception Handling, vol.26, no.10, pp. 978-1005, october 2000.

[Lieberherr01] Lieberherr, Karl, D. Orleans and J. Ovlinger, "Aspect- Oriented Programming with Adaptive Methods", Comunications of the ACM, vol. 44, no.10, pp. 29-32, october, 2001.

[Lippert01] Lippert, Martin and Stefan Roock. "Adapting XP to Complex Application Domains", ACM SIGSOFT Software Engineering Notes, Proceedings of the 8th European Software Engineering Conference, vol.26, pp. 316-317, september, 2001.

[Luporini85] Luporini, C. E. M. e M. N. Pinto, "Sistemas Administrativos", Atlas, SP,1985.

[Mann89] Mann, Nancy R., "The Keys of Excellence - The story of the Deming philosophy", PrestwickBooks, 1989.

[Martin87] Martin, James, "Recommended Diagramming Standards for Analysts and programmers: A Basis for Automation", Englewood Cliffs, NJ, Prentice-Hall, 1987.

[Martin88] Martin, James and Carma McClure, "Structured Techniques: A Basis for CASE", Englewood Cliffs, NJ, Prentice-Hall, 1988.

[Martin92] Martin, James and Odell, James J., "Object-Oriented Analysis and Design", Englewood Cliffs, NJ, Prentice-Hall, 1992.

[Martins99] Martins L. E. G. and Beatriz M. Daltrini, "An Approach to Software Requirements Elicitation Using Precepts from Activity Theory", Proceedings of the 14th International Conference on Automated Software Engineering, IEEE Computer Society Press,1999.

[Masiero92] Masiero, P. C., "Análise Estruturada de Sistemas pelo Método de Jackson", Editora Edgard Blucher Ltda, SP,1992

[Mayer95] Mayer, Richard J. and Christopher Menzel, "Information integration for concurrent engineering (LICE) IDEF3 process description capture method report", Knowledge Based Systems, Incorporated, 205p, Texas, 1995.

[McMenamim84] McMenamim Sthephen M., e Palmer, John F., "Essential Systems Analysis", Englewood Cliffs, NJ, Prentice-Hall, 1984.

[Melendez90] Melendez Filho, Rubem, "Prototipação de Sistemas de Informação", Livros Técnicos e Científicos, RJ, 1990.

[Mellor88] Mellor, J. Stephen e Shlaer Sally, "Object-Oriented System Analysis - Modeling the World in Data", Englewood Cliffs, NJ, Prentice-Hall, 1988.

[Mellor92] Mellor, J. Stephen e Shlaer Sally, "Object Lifecycles: Modeling the World in States", Englewood Cliffs, NJ, Prentice-Hall, 1992.

[Moore01] Moore, J. M. and Frank M. Shipman III., "Requirements Elicitation using Visual and Textual Information",IEEE, Proceedings of the Fifth International Symposium on Requirements Engineering RE'01, pp.307-309, Toronto, Canada, august, 2001.

[Muller01] Muller, Mathias M. and Walter F. Tichy, "Case Study: Extreme Programming in a University Environment.", Proceedings of the 23rd International Conference on Software Engineering, pp. 537-544,Toronto, Ontario, Canada, july, 2001.

[Nagappan03] Nagappan, Nachiappan, Laurie Williams, Miriam Ferzli, Eric Wiebe, Kai Yang, Carol Miller and Suzanne Balik, "Improving the CS1 Experience with Pair Programming.", ACM, Proceedings of the 34th technical symposium on Computer science education, Reno, Navada, pp. 359-362, february, 2003.

[Nassi73] Nassi, I. and B. Shneiderman, "Flowchart techiques for estructured programming", SIGPLAN Notices, august, 1973.

[Newkirk02] Newkirk, James, "Introduction to Agile Processes and Extreme Programming.", ACM, Proceedings of the 24th international conference on Software engineering, Orlando, Florida, pp. 695-696, may, 2002.

[Nogueira87] Nogueira, R. e Garcia, J., "Avaliação e Seleção de Sistemas", Livros Técnicos e Científicos, RJ, 1987.

[Nunes00] Nunes, Eduardo, " Visual Basic 6.0 - Guia do Usuário", Makron Books do Brasil, SP, 2000.

[Ntafos88] Ntafos, S. C., "A Comparison of Some Structural Testing Strategies", IEEE, Transaction on Software Engineering, vol. 14, no. 6, pp. 868-874, june, 1988.

[Olson01] Olson D., Hoh In and Tom Rodgers, "A Requirement Negotiation Model Based on Multi-Criteria Analysis", IEEE, Proceedings of the Fifth International Symposium on Requirements Engineering, RE'01, pp. 312-313, Toronto, Canada, august, 2001.

[Ossher01] Ossher, Harold (panelist), Tzilla Elrad (moderator), "Discussing Aspects of AOP", Comunications of the ACM, vol. 44, no.10, pp. 29-32, october, 2001.

[Penna00] Penna, Manoel Camillo, Sergio Mainetti Jr. e Vinicius Vendrami Malucelli, "O Mercado para Projetos com Objetos Distribuidos", Developers' CIO Magazine, número 43, Grande ABC Editora Gráfica, SP, março, 2000.

[Picarelli02] Picarelli, J. E. and Beatriz M. Daltrini, "Domain Analysis – A Technique for Requirements Analysis", Proceedings of the Sixth IASTED International Conference on Software Engineering and Applications, pp. 106-111, Cambridge, Mass., november, 2002.

[Pirsig84] Pirsig, Robert M., "Zen e a Arte de Manutenção de Motocicletas - Uma investigação sobre valores", Paz e Terra S/A, RJ, 1984.

[Pressman92] Pressman, Roger S., "Software Engineering: A Practitioner's Approach", Third Edition, McGraw-Hill, 1992.

[Pressman97] Pressman, Roger S., "Software Engineering: A Practitioner's Approach", Fourth Edition, McGraw-Hill, 1997.

[Procaccino02] Procaccino, J. Drew, June M. Verner, Scott Overmyer and Marvin Darter, Marvin, "Case study: factors for early prediction of software development sucess", Information on Software Technology, vol. 44, pp. 53-63, 2002.

[Quatrani00] Quatrani, Terry, "Visual Modeling with Rational Rose 2000 and UML", Addison- Wesley, 2000.

[Rashid02] Rashid Awais, Peter Sawyer, Ana Moreira and João Araújo, "Early Aspects: a Model for Aspect-Oriented Requirements Engineering", Proceedings of the IEEE Joint International Conference on Requirements Engineering, RE'02, pp.199-202, Essen, Germany, september, 2002.

[Renaud93] Renaud, Paul E., "Introduction to Client/Server Systems", John Wiley & Sons, 1993.

[Riemensch02] Riemenschneider, Cyntia K., Bill C. Hardgrave and Fred D. Davis, "Explaining Software Developer Acceptance of Methodologies: A Comparison of Five Theoretical Models", IEEE Transaction on Software Engineering, vol. 28, no.12, pp. 1135-1145, Essen, Germany, december, 2002.

[Rocha87] Rocha, A. R. C., "Análise e Projeto Estruturado de Sistemas",Campus, RJ, 1987.

[Rolland01] Rolland C. and Naveen Prakash, "Matching ERP System Functionality to Customer Requirements", IEEE, Proceedings of the Fifth International Symposium on Requirements Engineering, RE'01, pp. 66-76, Toronto, Canada, august, 2001.

[Ross84] Ross, D., "Applications and Extensions of SADT", IEEE Computer, vol.18, no. 4, pp. 25-35, april, 1984.

[Ross88] Ross, Phillip J., "Taguchi Techniques for Quality Engineering", McGraw-Hill, 1988.

[Rumbaugh91] Rumbaugh, James, "Object-Oriented Modeling and Design", Englewood Cliffs, NJ, Prentice-Hall, 1991.

[Saffo93] Saffo, Paul, "Reflexões para o Futuro - Com a Palavra", Tradução de Angela Pimenta e Lígia Paula Silber, Editora Abril S/A, SP, 1993.

[Saviani92] Saviani, J. R., "O Analista de Negócios e da Informação", Atlas, SP,1992.

[Scheer00] Scheer, August Wilhelm & Habermann Frank, "Using business process models to achieve positive results", Communication of the ACM, vol. 43, no. 4, 2000.

[Scholtes88] Scholtes, Peter R., "The team handbook", Joiner Associates, 1988.

[Schrefl02] Schrefl, M. and M. Stumptner, "Behavior-Consistent Specialization of Object Life Cycles", ACM, Transactions on Software Engineering and Methodology, TOSEM'02, vol.11, no. , pp.98-148, 2002.

[Sebesta99] Sebesta, Robert W., "Concepts of programming languages", Addison-Wesley, 1999.

[Seilheimer00] Seilheimer, Steven D., "Information management during systems development: a model for improvement in productivity", International Journal of Information Management, Cambridge, vol.20, no.4, pp.287-295, august, 2000.

[Shiller90] Shiller, Larry, "Software Excellence", Englewood Cliffs, NJ, Prentice-Hall, 1990.

[Silva02] Silva Andrés, "Requirements, Domain and Specifications: A Viewpoint-based Approach to Requirements Engineering", ACM, Proceedings of 24th International Conference on Software Engineering, ICSE'02, pp. 94-104, Orlando, Flórida, USA, may 2002.

[Simon69] Simon, H., "The Sciences of the Artificial", Cambridge, MIT Press, 1969.

[Smith02] Smith, R. L., Avrunin, G. S., Clarke, L.A. and Osterweil L. J., "PROPEL: An Approach Supporting Property Elucidation", ACM, Proceedings of 24th International Conference on Software Engineering, ICSE'02, pp. 11-21, Orlando, Flórida, USA, may 2002.

[Sommerville97] Sommerville, Ian and Pete Sawyer, "Requirements Engineering", John Wiley & Sons, 1997.

[Sommerville01] Sommerville, “Software Engineering”, Addison-Wesley, 2001.

[Souther77] Souther, James & Wite, Myron L., “Technical Report Writing”, John Wiley & Sons, 1977.

[Stallings96] Stallings, Willian, “Computer Organization and Architecture”, Englewood Cliffs, NJ, Prentice-Hall, 1996.

[Stevens81] Stevens, Wayne Paul, “Using Structured Design”, John Wiley & Sons, 1981.

[Szmit93] Szmit, R., “Programação Orientada para Objeto”, Livros Técnicos e Científicos, RJ, 1993.

[Takahashi90] Takahashi, Tadao, Hans K. E. Leisemberg e Daniel Tavares Xavier, “Programação Orientada a Objetos - Uma Visão Integrada do Paradigma de Objetos”, VII Escola Computação, IME-USP, SP, 1990.

[Thayer97] Thayer, R. H. and Dorfman, M., “Introduction to Tutorial Software Requirements Engineering”, Software Requirements Engineering, 2nd Ed. IEEE, 1997, pp 1-2.

[Toigo89] Toigo, Jon Willian, “Disaster Recovery Planning”, Englewood Cliffs, NJ, Prentice-Hall, 1989.

[Trilnik02] Trilnik, Federico, A. D. Pace and M. Campo, “Smartweaver: An Agent-Based Approach for Aspect-Oriented Development”, ACM, Proceedings of 24th International Conference on Software Engineering, ICSE’02, pp. 716-717, Orlando, Flórida, USA, may 2002.

[Umbaugh89] Umbaugh, Robert E., “Establishing effective steering committees.”, Information Management, pp. 1-10, 1989.

[Vickers02] Vickers A., Alistair Mavin and Helen May, "Requirements Engineering: How do you know how good you are ?", Proceedings of the IEEE Joint International Conference on Requirements Engineering, RE'02, pp.194-195, Essen, Germany, september, 2002.

[Walton89] Walton, Richard E., "Up and running: integrating information technology and the organization", Harvard Business Schol Press, 1989.

[Ward85] Ward, Paul T. and S. J. Mellor, "Structured Development for Real-Time Systems", Yourdon Press, 1985.

[Ward86] Ward, Paul T., "Systems Development Without Pain", Englewood Cliffs, NJ, Prentice-Hall, 1986.

[Warnier85] Warnier, J. D., "Guia dos Usuários de Sistemas de Informação", Campus, RJ, 1985.

[Warnier84] Warnier, J. D., "LCS - Lógica de Construção de Sistemas", Campus, RJ, 1984.

[Weinberg75] Weinberg, Gerald M., "In Introduction to General Systems Thinking", John Wiley & Sons, NY, 1975.

[Weinberg85] Weinberg, Gerald M., "The Secrets of Consulting", McGraw-Hill, 1985.

[Weinberg88] Weinberg, Gerald M., "Rethinking Systems Analysis and Design", McGraw-Hill, 1988.

[Weinberg92] Weinberg, Gerald M., "Quality Software Management", Dorset House, 1992.

[Weinberg93] Weinberg, Gerald M., "Quality Software Management : First-Order Measurement", Dorset House, 1993.

[Weinberg94] Weinberg, Gerald M., "Quality Software Management : Congruent Action", Dorset House, 1994.

[Weiss02] Weiss, David M., David Bennet, John Y. Payseur, Pat Tendick and Ping Zhang, "Goal-Oriented Software Assesment", ACM, Proceedings of 24th International Conference on Software Engineering, ICSE'02, pp. 221-231, Orlando, Flórida, USA, may 2002.

[Wessels01] Wessels, Bridgette and John Dobson, "What Happens before Requirements Engineering?", IEEE, Proceedings of the Fifth International Symposium on Requirements Engineering RE'01, pp. 298-299, Toronto, Canada, august, 2001.

[Weyuker86] Weyuker, E. J., "Axiomatizing Software Test Data Adequacy", IEEE Transaction on Software Engineering, december, 1986.

[Whitmire95] Whitmire, S. A., "An Introduction to 3D Function Points", Software Development, April,1995.

[Wiener50] Wiener, Norbert, "Cibernética e Sociedade - O uso humano de seres humanos", Cultrix, SP,1950 (1993).

[Wieringa98] Wieringa, R., "A Survey of Structured and Object-Oriented Software Specification Methods and Techiniques", ACM Computing Surveys, vol. 30, no. 4, december, 1998.

[Williams00] Williams, Laurie A. and Robert R. Kessler, "All I Really need to Know About Pair Programming I Learned In Kindergarten", Communications of the ACM, vol. 43, no. 5, pp. 108-114, may, 2000.

[Williams01] Williams, Laurie A., "Integrating Pair Programming into a Software Development Process.", IEEE, Proceedings of the 14th. Conference on Software Engineering Education and Training, CSEET'01, pp. 27-36, Charlote, North Carolina, february, 2001.

[Wimblad90] Winblad, Ann L., "Object-Oriented Software", Addison-Wesley, 1990.

[Witehead71] Witehead, A. N., "Os Fins da Educação e outros Ensaios", Cultrix, SP, 1971.

[Woodside02] Woodside M., D. Petriu and K. Siddiqui, "Performance-related Completions for Software Specifications", ACM, Proceedings of 24th International Conference on Software Engineering, ICSE'02, pp. 22-32, Orlando, Flórida, USA, may 2002.

[Worf56] Worf, B., "Language, Thought and Reality", MIT Press, 1956.

[Yourdon85] Yourdon, Edward, "Structured Walktroughs", Englewood Cliffs, NJ, Prentice-Hall, 1985.

[Yourdon86] Yourdon, Edward, "Managing the Structured Techniques", Englewood Cliffs, NJ, Prentice-Hall, 1986.

[Yourdon88] Yourdon, Edward, "Managing the System Life Cycle", Englewood Cliffs, NJ, Prentice-Hall, 1988.

[Yourdon89] Yourdon, Edward, "Modern Structured Analysis", Englewood Cliffs, NJ, Prentice-Hall, 1989.

[Yourdon92] Yourdon, Edward, "Decline & Fall of the American Programmer", Englewood Cliffs, NJ, Prentice-Hall, 1992.

[Yourdon96] Yourdon, Edward, "Rise & Resurrection of the American Programmer", Englewood Cliffs, NJ, Prentice-Hall, 1996.

[Zhang02] Zhang, Qian and Armin Eberlein, "Deploying Good Practices in Different Requirements Process Models", Proceedings of the Sixth IASTED International Conference on Software Engineering and Applications, pp.76-81, november, 2002.

[Zowghi01] Zowghi, D., V. Gervasi and McRae., "Using Default Reasoning to Discover Inconsistencies in Natural Language Requirements", IEEE, Proceedings of the Eighth Asia-Pacific Software Engineering Conference, APSEC'01, pp.133-140, 2001.

ANEXO I - Estudo de caso

. Introdução

Trata-se da análise de requisitos uma aplicação de software para viabilização e realização de eleições genéricas. O termo genérica significa que o software deve suportar a realização completa de qualquer tipo de eleição, de julgamento, de plebiscito, de coleta e apuração de preferências, de opiniões entre outras.

. Justificativas

Esta aplicação pode ser também um estudo de caso para a utilização, além da *Análise por Visões*, de outras técnicas e/ou metodologias de desenvolvimento de software. Possui aparentemente um grau de complexidade entre pequena e média. Isto pode permitir que a sua modelagem e implantação seja efetuada em um prazo relativamente curto, de seis a dez meses, contando com o trabalho de apenas um analista em tempo integral.

Comercialmente, pode ter seu desenvolvimento justificado por meio da venda do produto para escolas, sindicatos, clubes, prefeituras, estabelecimentos comerciais e outros tipos de organizações. Pode também ser um produto, o software obtido via *download,* oferecido grátis, aos clientes de um provedor. Pode ser suporte para um serviço, as eleições em si, oferecido por algum provedor. Pode ser oferecido, através da mídia CD, como brinde e publicidade de políticos quando candidatos, etc. No âmbito da computação pessoal pode ser utilizada como entretenimento. Aparentemente, é grande o potencial comercial desta aplicação.

Tecnicamente, a aplicação abrange vários aspectos. Envolve vários tipos de usuários e várias funcionalidades. Algumas atividades devem ser centralizadas e outras distribuídas. Para sua implementação devem ser utilizadas tecnologias de banco de dados, processamentos tipo *batch* e *transaction, off-line* e *on-line.*

. Abertura do projeto

Decidido o desenvolvimento da aplicação faz-se a abertura de um projeto. Isto é feito por meio da geração de uma instância da classe *Projeto.* Essa instanciação é efetuada, conforme proposto na Tabela 2.3, na forma de a*tribuição*. Tem-se então:

PROJETO

Referência-Projeto	: EC (de **E**studo de **C**aso)
Nome-Projeto	: SEG- Software de Eleição Genérica
Nome-Fantasia-Projeto	: Eleição - 1.0
Resumo-Tema	: Viabilizar o preparo, a configuração e realização de qualquer tipo de eleição.
Descrição-Tema	: Aplicação de software que oferece alternativas para configurar e realizar, da organização até a apuração, de eleições, pleitos, plebiscitos de qualquer modalidade.
Data-prevista-início	: 15/09/2002
Data-real	: 16/10/2002
Data-prevista-final	: 30/12/2003
Data-real-final	: 13/01/2003 - Análise de Requisitos
Observação	: A implementação tem início previsto para 13/03/2003.

. Início do projeto

Como na grande maioria das aplicações de software, os desejos e necessidades do que devem ou não ser contemplados por uma aplicação começam a ser explicitados de forma desordenada, não claros, incompletos, mal redigidos, ou até mesmo na forma oral. São às vezes jogados no papel através de um modelo descritivo. Neste estudo de caso, o primeiro modelo descritivo consta na *Descrição Geral*, elaborado da forma de *descrição* (Tabela 2.3) apresentado em seguida.

. Descrição Geral

Esta aplicação de software *deve ter aplicabilidade em qualquer tipo de pleito, como por exemplo, concurso de beleza, questões propostas em assembléias de sindicatos, centros acadêmicos, coleta de grau de satisfação de clientes de supermercado, plebiscitos rápidos, eleições partidárias, vereadores/ prefeitos/ deputados/ senadores /governadores/ presidente.*

Deve ser, em sua concepção e implantação, suficientemente flexível, possível de adaptar os diversos tipos de eleições e de atender também a tipos de eleições não previstas. Deve ter as opções de funcionamento: totalmente isolada em um PC, integrada em redes de PCs e na Web. Deve ter o nível de segurança desejado para cada eleição.

Para cada funcionalidade oferecida deve ter várias opções de interfaces. Esta será escolhida pelo usuário da mesma. Pode-se ter usuário analfabeto. Deve permitir processos de auditoria por fiscais pré-cadastrados, processos de rastreamento (traces), de cadastramento de registros de históricos, de ajustes e acertos e processo que possibilitem comparações entre eleições. Deve possibilitar a utilização de cadastros entre uma eleição e outra e também o compartilhamento de cadastros.

A análise de requisitos pode contar com um esforço de apenas 15 horas semanais de análise de um analista Sr. durante apenas 4 semanas. A implementação e implantação é desejada porém não obrigatória. Interessante seria estar, total ou parcialmente pronta até o final do ano de 2002. O parcialmente pronta, versão 1.0, deve contemplar no mínimo todas as funcionalidades de uma eleição simples. Porém, os recursos para implementação, tais como programação, se não alocados, devem ser, no mínimo, planejados. O desenvolvimento de novas versões (releases), desejável até o 3.0, deve também ser planejado em termos de requisitos a contemplar.

. Descrição Geral (continuação)

Deve possibilitar a convocação automática de eleitores e operadores, opcionalmente com ou sem garantia do recebimento desta convocação, através de emissão e/ou controle de protocolo. Deve possibilitar também o cadastramento e habilitação de eleitores e de candidatos via e-mail.

Conforme o tipo de eleição, o eleitor poderá votar mais de uma vez, no mesmo cargo ou em cargo diferentes, sendo a urna liberada, por controle exterior, para o mesmo eleitor. Se um eleitor demorar muito para votar a urna deverá ser inibida automaticamente.

As dúvidas, esclarecimentos e ajudas (helps) devem ter disponibilidade rápida e ser o mais auto-explicativas possíveis para que com o auxílio ad hoc necessário seja minimizado. As atividades ad hoc de cadastramento e atribuição de senhas não podem ser feitas por eleitores e candidatos.

. Tabela de Necessidades

A partir do(s) modelo(s) descritivo(s) e/ou de declarações explicitadas, mesmo as de forma oral, pode-se iniciar o preenchimento de uma *Tabela de Necessidades*.

Não há uma ordem ou critério pré-estabelecido para o preenchimento da mesma. A referência de valor *01* não significa que este requisito é o primeiro a ser contemplado ou qualquer outro significado subliminar.

Há uma interpretação e transformação da declaração de necessidade em um ou mais requisitos implementáveis. Por exemplo de *aceitar vários pleitos* para *configurar eleiçã*o e *possibilitar vários tipos de voto e de candidato.* Também já se caracteriza o requisito/restrição por sua *Natureza-Necessidade* e por *Tipo-Plano* conforme proposto no modelo referência da *Análise por Visões* da Figura 3.1. Agrupar os requisitos/restrições futuramente por *Tipo-Plano* pode ser útil para a escolha da ferramenta e a efetiva modelagem.

.Tabela de Necessidades – Estudo de Caso/ Sistema de Eleição Genérica

Ref	Declaração	**Requisito / Restrição**	**Natureza**	**Plano**
01	*Aceitar qualquer tipo de pleito*	Parametrizar entidades para configurar eleição	Funcional	Utilização
		Possibilitar vários tipos de voto e de candidato	Funcional	Utilização
02	*Suficientemente flexível - suportar futuros tipos eleições*	Facilidade de manutenção / evolução	Não funcional	Software
03	*Funcionar isolado em "PC" e em rede*	Processamento em computador pessoal e em Intranet / Internet, arquitetura C/S.	Não funcional	Produção
04	*Ter segurança*	Organizar geração e disponibilização das informações em qualquer mídia	Não funcional	Produção
		Estabelecer procedimentos para *back-ups e up-grades.*	Não Funcional	Produção
		Estabelecer cadastramento e níveis de autorizações (senhas)	Não Funcional	Produção

. Tabela de Necessidades – Estudo de Caso/ Sistema de Eleição Genérica (continuação)

Ref	Declaração	Requisito / Restrição	Natureza	Plano
05	*Interfaces para vários níveis de eleitores (analfabetos)*	Oferecer várias opções de interface	Funcional	Utilização
		Ter grau elevado de usabilidade e facilidade operacional	Não Funcional	Software
06	*Auditoria e fiscalização*	Criar históricos	Não funcional	Produção
07	*Permitir cadastramento de traces*	Criar "logs"	Não Funcional	Produção
08	*compartilhamento de cadastros*	Não vincular as entidades (cadastros) a uma eleição específica	Funcional	Utilização
09	*15 homen/hora semanais para atividade de análise*	Planejar realisticamente	Não funcional	Administração
10	*Produtos conforme Release 1.0, 2.0, 3.0*	Planejar por versões (*release)*	Não Funcional	Administração
11	*Análise final de dezembro 2002, outras depois*	Planejar implementação	Não funcional	Administração
12	*Planejar implementação*	Planejar tecnologia	Não funcional	Produção
13	*Convocar eleitores e operadores*	Possibilitar o endereçamento de mensagens	Funcional	Utilização
14	*Cadastramento via e-1/2 de eleitores e candidatos*	Possibilitar cadastramento de forma remota (e-1/2, planilha-tela, ..)	Não Funcional	Produção
15	*Conforme tipo de eleição o eleitor pode votar mais vezes*	Identificar unicamente o eleitor	Funcional	Utilização
		Contabilizar voto por eleitor	Funcional	Utilização
16	*A urna deve ser liberada ad hoc*	Controlar abertura /fechamento por voto / eleitor	Funcional	Utilização
		Oferecer tela para abertura/ fechamento urna pelo Presidente Mesa	Funcional	Software
17	*A urna deve ser inibida por time out*	Estabelecer *time-out* para tela de votação	Não funcional	Produção
18	*Inibir (qualquer função) três tentativas fracassada de senha*	Estabelecer controle de acesso	Não funcional	Produção
19	*Otimizar atendimento de dúvidas / esclarecimento*	Estabelcer esquema de FAQ	Funcional	Utilização
20	*Eleitor e candidato não podem ser operadores*	Controlar estados de candidatos / eleitores e operadores	Funcional	Utilização

. Tabela de Necessidades – Estudo de Caso/ Sistema de Eleição Genérica (continuação)

Ref	Declaração	Requisito / Restrição	Natureza	Plano
21	*Impugnação voto só pode ser feita pelo Juiz Eleitoral*	Possibilitar impugnação / cancelamento	Funcional	Utilização
		Elaborar tela para Juiz de forma *ad hoc* efetuar suas tarefas	Funcional	Software
22	*Impugnação da eleição só pode ser feita pelo Juiz*	Possibilitar impugnação / cancelamento	Funcional	Utilização
		Elaborar tela para Juiz de forma *ad hoc* efetuar suas tarefas	Funcional	Software
23	*Abertura / Encerramento da eleição / votação por Juiz*	Possibilitar abertura / encerramento	Funcional	Utilização
		Elaborar tela para Juiz de forma *ad hoc* efetuar suas tarefas	Funcional	Software
24				
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•
•		•	•	•
		•		•
				•

Notar que esta tabela não se esgota dentro de uma fase ou etapa. Ela deve ficar aberta, pois necessidades podem surgir independentemente de nossa vontade. Ela somente será encerrada após a finalização do desenvolvimento do projeto. Atualizações na tabela ainda podem ser feitas se um novo projeto, caracterizado como *manutenção* ou *evolução,* da mesma aplicação, for aberto.

. Determinação de Limite(s)

Tendo-se um contexto definido em *Projeto*, pode-se então determinar o *Limite*, conforme modelo referência da *Análise por Visões* da Figura 3.1.

Realiza-se neste ponto, para este estudo de caso, a primeira atividade de análise propriamente dita. Análise no sentido de particionar um contexto. É uma atividade baseada em experiências, vinculada totalmente na perspectiva e habilidades do analista. Cada modelo sai segundo a visão de cada modelador. Por não possuir maneiras, regras ou critérios pré-estabelecidos torna-se uma tarefa crítica, bastante delicada e arriscada. O domínio foi aqui particionado tendo uma referência cronológica. O *Limite* foi então particionado, para efeito de modelagem e representação, nos três *Limites, L1, L2 e L3* :

LIMITE

Referência-Limite : L0
Descrição-Limite : Eleição Genérica – (o todo)
Declaração-Objetivo : Preparar eleição, realizar votação e apurar eleição.
Observação : ------
Abordagem : Sistema
Estratégia : *top-down*
Tipo-representação : Gráfica
Representação : DLS L1, L2, L3.

.VISÃO 00

Como toda representação exige uma *Visão*, para o(s) *Limite*(s) tem-se para o todo e valendo para as partes, a visão *V00*:

Referência-Visão : V00
Referência-Limite : L0
Referência-Foco : F1
Referência-Plano : P1
Observação : -----

Referência-Foco : F1
Tipo-Foco : Dado
Observação : -----

Referência-Plano : P1
Tipo-Necessidade : Requisito
Natureza-Necessidade: Funcional
Tipo-Plano : Utilização

A seguir as definições e representações de cada *Limite L1, L2* e *L3*.

Referência-Limite : L1

Descrição-Limite : Preparação da Eleição

Declaração-Objetivo : Levantar o tipo de eleição. Configurar. Efetuar o cadastramento necessário.

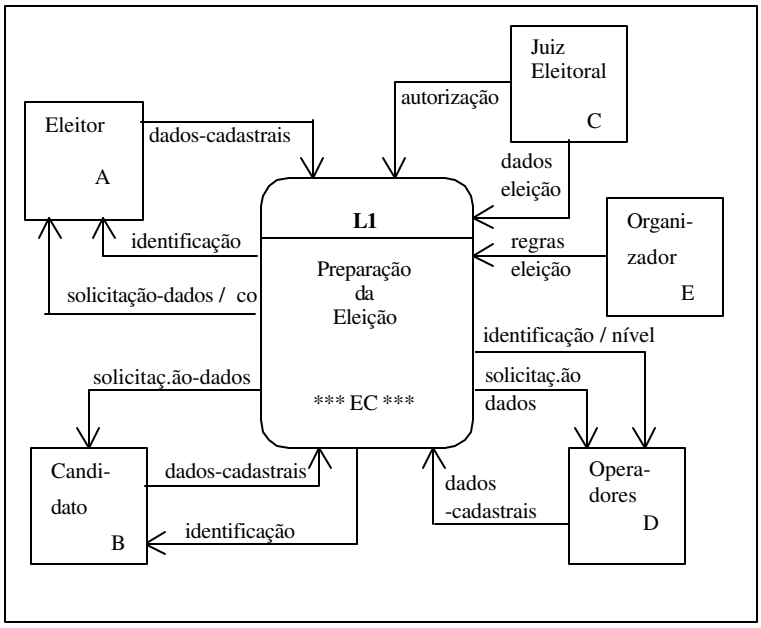
Observação : ----

Abordagem : Sistema

Estratégia : *top-down*

Tipo-representação : Gráfica

Representação : DLS - Preparação da Eleição



Referência-Limite : **L2**

Descrição-Limite : Realização da Votação

Declaração-Objetivo : Viabilizar e realizar a votação.

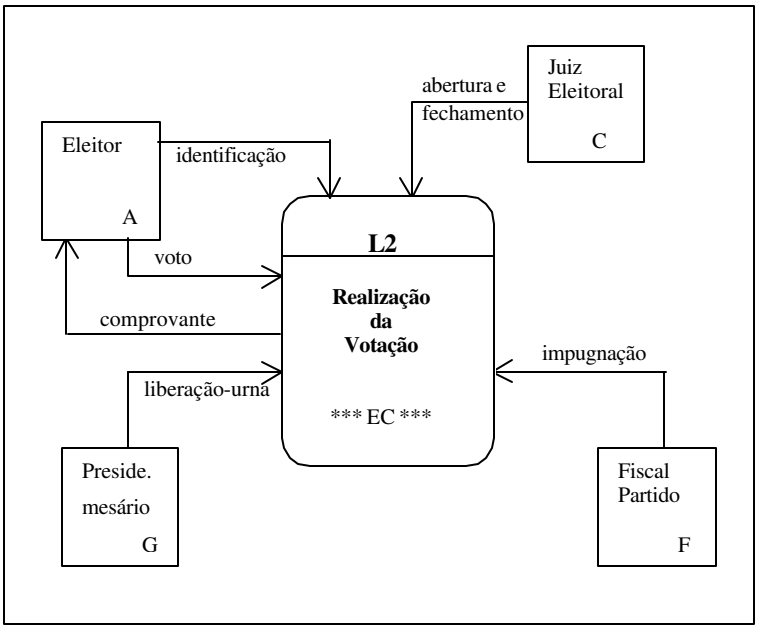
Observação : ----

Abordagem : Sistema

Estratégia : *top-down*

Tipo-representação : Gráfica

Representação : DLS - Realização da Votação



Referência-Limite : **L3**

Descrição-Limite : Apuração da Eleição

Declaração-Objetivo : Apurar a votação e disponibilizar resultados.

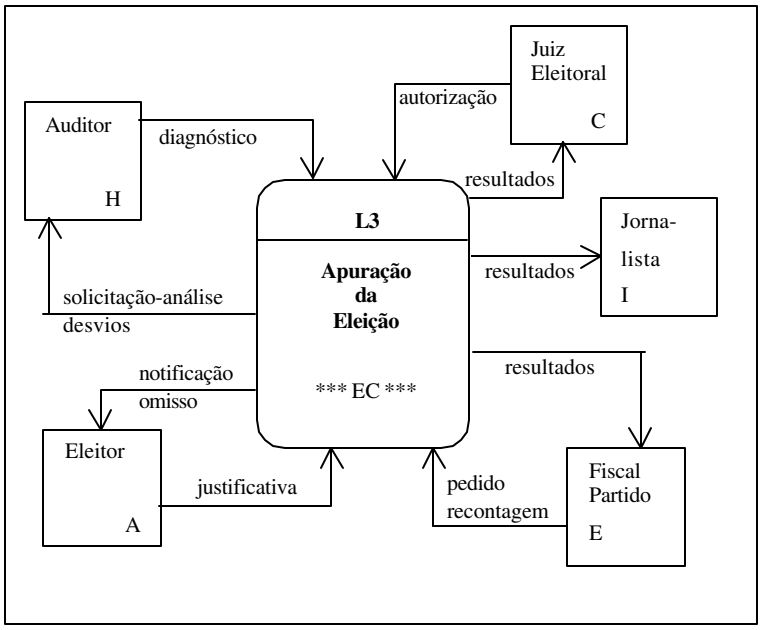
Observação : ----

Abordagem : Sistema

Estratégia : *top-down*

Tipo-representação : Gráfica

Representação : DLS - Apuração da Eleição



. Determinação de Visão(ões)

Tendo como referência o modelo referência da *Análise por Visões* apresentado na Figura 3.1, a análise e/ou o detalhamento pode ser efetuada através das determinações das *Visões*. A determinação de cada *visão* não obedece a uma ordem geral pré-determinada. Conforme se levantam as informações sobre a aplicação, uma *visão* pode ser determinada e modelada. A *Tabela de Visões* aqui apresentada foi sendo preenchida no decorrer da análise de requisitos. Não foi elaborada de uma só vez, não foi efetuada uma análise geral e pré-definidas as *visões*. As *visões* foram surgindo sucessivamente. Esta tabela serve para efetuar o controle de referências das várias *visões* e serve também como um tipo de índice .

Tabela de Visões

Referência	**Descrição**
V00	DLS - Limites L1, L2, L3
V01	DCU - Eleição Genérica
V02	Relação de Eventos
V03	DRE - Reação Eventos 1 a 20
V04	DRE - *Time-out* da urna
V05	DAC - Eleição Genérica (utilização)
V06	DCL - Eleição Genérica (utilização)
V07	DAC - Eleição Genérica (software)
V08	DCL - Eleição Genérica (software)
V09	DES - Constituir Eleitor
V10	DBL - Cadastrar Eleitor
V11	Planilhas papel
V12	Relação tecnologia prevista
V13	Diagrama plataforma J2EE
V14	DD – Elementos de Dados
V15	DD – Estrutura de dados
V16	*DNL - Especificação função*
V17	*Cronograma por atividades*
V18	*Cronograma por produtos*
V19	
.	.
.	.
.	.

.VISÃO 01

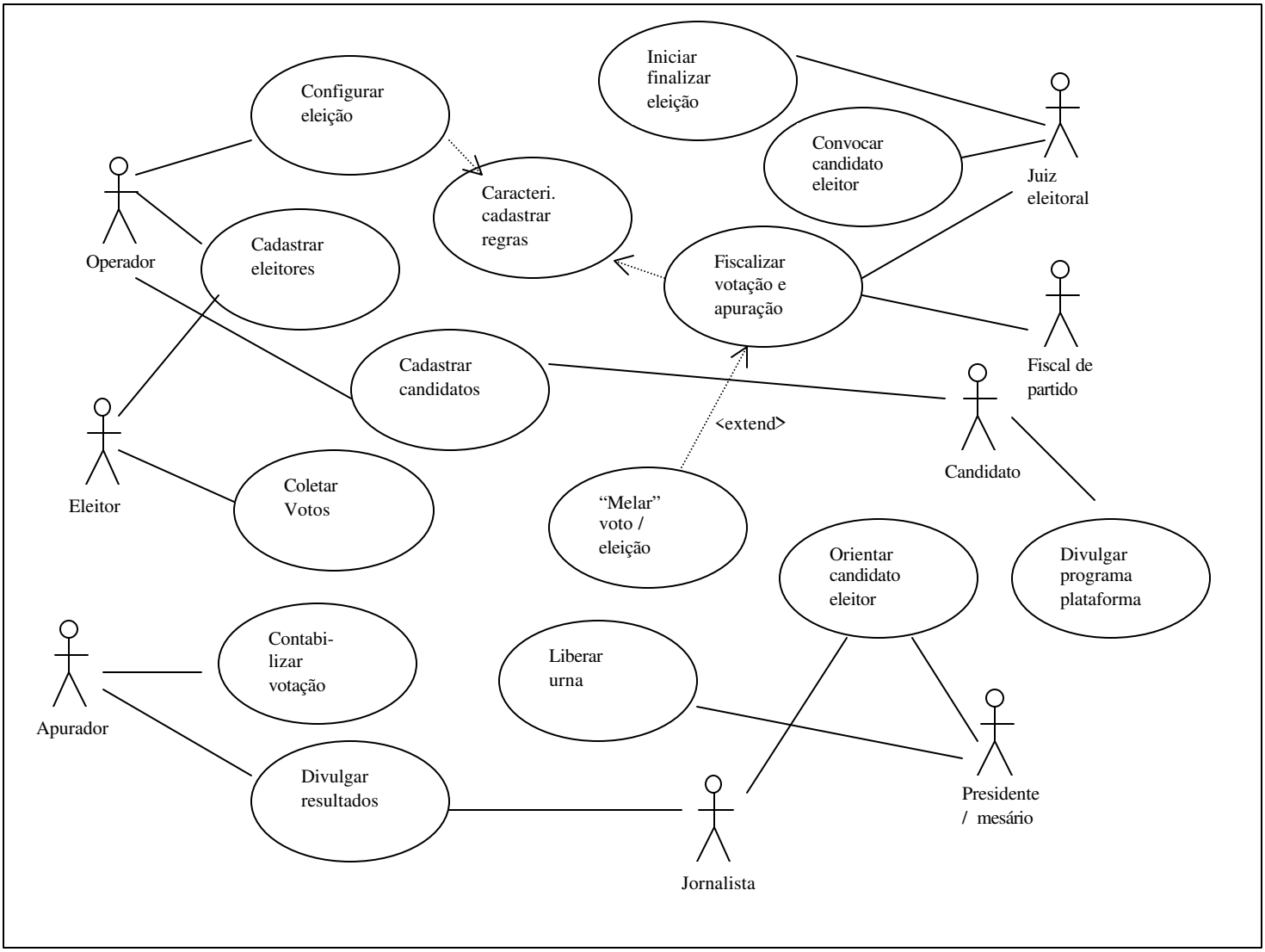
Esta visão foi determinada a partir da visão de referência 00. A escolha pelo DCU, veio do fato de que uma vez determinada as *Entidades Externas*, ficou, no caso, fácil determinarem os atores. Elaborar um DFD de contexto único, a partir dos três diagramas de contextos de cada *Limite* seria esteticamente não recomendado. Efetuar uma *explosão*, da *Eleição Geral*, por meio de um DFD nível 1,obtendo-se as macrofunções do sistema, a integração via memória e suas possíveis interações, além de mais difícil e demorado, o modelo gerado, poderia ficar mais complexo. Assim, apresentar apenas as macrofunções visíveis, neste momento, foi considerada adequada.

Referência-Visão : V01
Referência-Limite : L0
Referência-Foco : F1
Referência-Plano : P1
Observação : -----

Referência-Foco : F2
Tipo-Foco : Função
Abordagem : Sistema
Estratégia : *Top-down*
Observação : -----
Tipo-representação : Gráfica
Representação : DCU - Eleição Geral

Referência-Plano : P1
Tipo-Necessidade : Requisito
Natureza-Necessidade: Funcional
Tipo-Plano : Utilização

EC - V01 - DCU - Eleição Genérica



.VISÃO 02

Esta visão foi determinada a partir da visão de referência 01. A escolha foi pela *Relação de Eventos*, na forma de *Relação* (Tabela 2.3) obtida a partir da *Análise de Eventos* (item 3.6). Uma vez determinado os atores, ficou, no caso, fácil determinar quais são realmente os agentes externos (*Eleitor* e *Candidato*) e quais são agentes *ad hoc* (os demais) do sistema. Para cada um deles, foram então levantados os eventos que eles provocam, ou influenciam dando apoio.

Referência-Visão : V02
Referência-Limite : L0
Referência-Foco : F4
Referência-Plano : P1
Observação : -----

Referência-Foco : F4
Tipo-Foco : Evento
Abordagem : Sistema
Estratégia : *bottom-up*
Observação : -----
Tipo-representação : Textual - tipo Relação
Representação : Relação Eventos - Eleição Genérica

Referência-Plano : P1
Tipo-Necessidade : Requisito
Natureza-Necessidade: Funcional
Tipo-Plano : Utilização

EC - V02 – Relação de Eventos - Eleição Genérica

Referência	**Descrição evento**	**Tipo**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
01	Eleitor solicita cadastramento	1	S	N	S	N	S	S	S	N
02	Eleitor solicita permissão para votar	1	S	N	S	N	S	S	N	S
03	Eleitor realiza voto	1	S	N	S	N	S	S	S	N
04	Eleitor solicita orientação	1	S	N	S	N	N	S	S	S
05	Candidato solicita cadastramento	1	S	N	S	N	N	S	S	S
06	Candidato solicita orientação	1	S	N	S	N	N	S	S	S
07	Momento de divulgar programa / plataforma	3	N	S	S	N	N	S	N	N
08	Operador solicita cadastramento	2	S	N	S	N	S	S	S	S
09	Operador solicita orientação	3	S	N	S	N	N	S	S	S
10	Momento de configurar a eleição	2	N	S	S	N	N	S	S	S
11	Fiscal partido “mela” voto	2	S	N	S	N	S	S	S	S
12	Juiz eleitoral cancela voto	2	S	N	S	N	S	S	S	S
13	Fiscal partido “mela” eleição	2	S	N	S	N	S	S	S	S
14	Juiz eleitoral cancela eleição	2	S	N	S	N	S	S	S	S
15	Momento de finalizar votação	3 ou 4	N	S	N	N	S	S	S	N
16	Momento de finalizar eleição	3 ou 4	N	S	N	N	S	S	S	N
17	Momento de apurar os votos	3	N	S	N	N	N	S	S	N
18	Momento de divulgar resultados	3	N	S	S	N	N	S	N	N
19	Fiscal partido solicita recontagem	2	S	N	S	N	N	S	S	S
20	Momento de inibir urna *(time-out)*	5	N	N	N	N	S	S	N	S

Tipo	**Agente provocador**
1	Explícito Planejado
2	Explícito *ad-hoc*
3	Tempo Imprevisível
4	Tempo Previsível
5	Ocorrência do nada

Item	**Perguntas (S/ N)**
1	Há a necessidade de um fluxo de dados representando o estímulo do evento ?
2	Há a necessidade de um fluxo de controle representando o estímulo do evento ?
3	Gera uma ou mais respostas ao ambiente ?
4	Há um estímulo da atividade fundamental para uma atividade custodial ?
5	Há mudança de estado de um ou mais objetos ?
6	Há a necessidade de consulta à memória do sistema ?
7	Há a necessidade de atualização da memória do sistema ?,
8	Há a necessidade de pedir/receber informações do interagente *ad-hoc* ?

Cada evento foi caracterizado segundo o tipo proposto na Tabela 3.3. Foram respondidas as 8 questões propostas, utilizadas como orientação para determinar as reações aos eventos. Os eventos de números 15 e 16 podem ser caracterizados de ambas as maneiras. Pode ser evento do tipo 3 ou 4, dependendo do desejo do cliente. Se for do tipo 4, a questão de número2 deverá ter como resposta um não (N).

.VISÃO 03

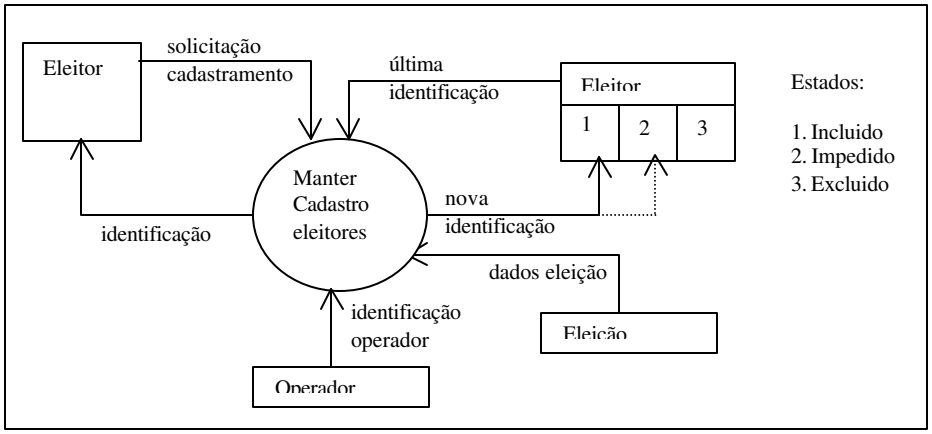
Tendo uma *Relação de Eventos* já caracterizados (visão de referência 02) e as respostas às oitos perguntas de orientação modela-se a reação para cada um dos eventos dessa relação. As atividades custodiais não foram modeladas separadas da fundamental, por facilidade de construção e clareza na apresentação dos modelos.

Referência-Visão : V03
Referência-Limite : L1
Referência-Foco : F2
Referência-Plano : P1

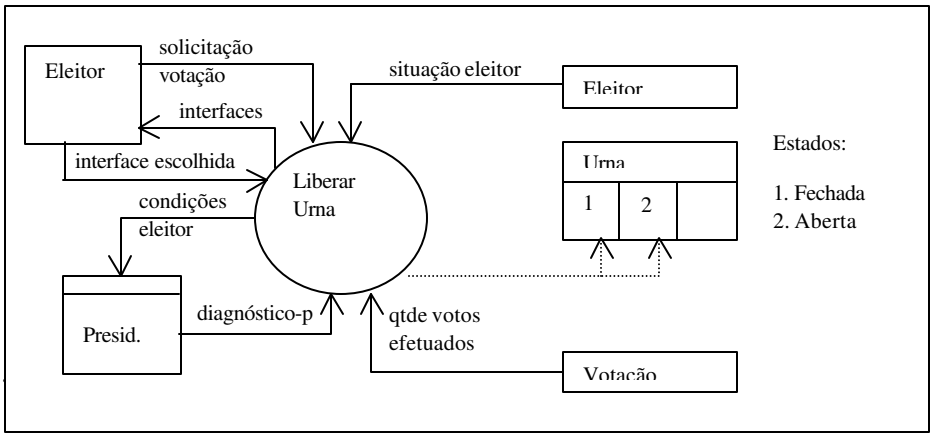
Referência-Foco : F2
Tipo-Foco : Função
Abordagem : Sistema
Estratégia : *bottom-up*
Tipo-representação : Gráfica
Representação : DRE - Reação a Eventos (01 a 20)

Referência-Plano : P1
Tipo-Necessidade : Requisito
Natureza-Necessidade: Funcional
Tipo-Plano : Utilização

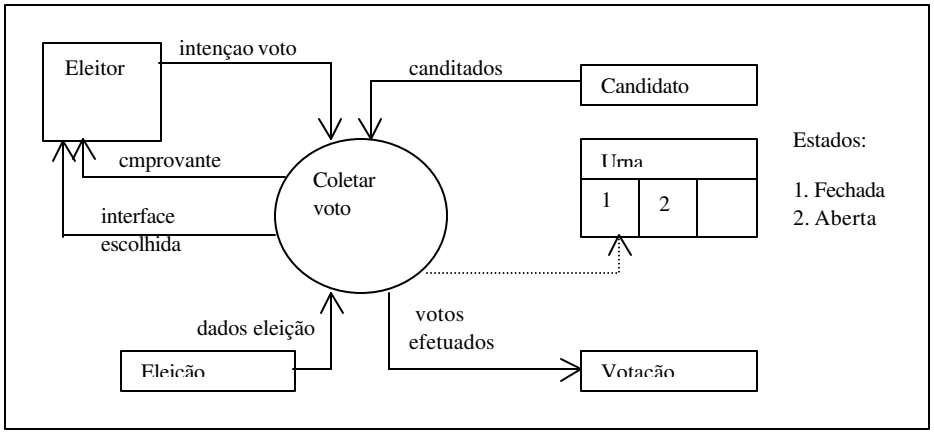
EC – V03 - DRE - 01. Eleitor solicita cadastramento



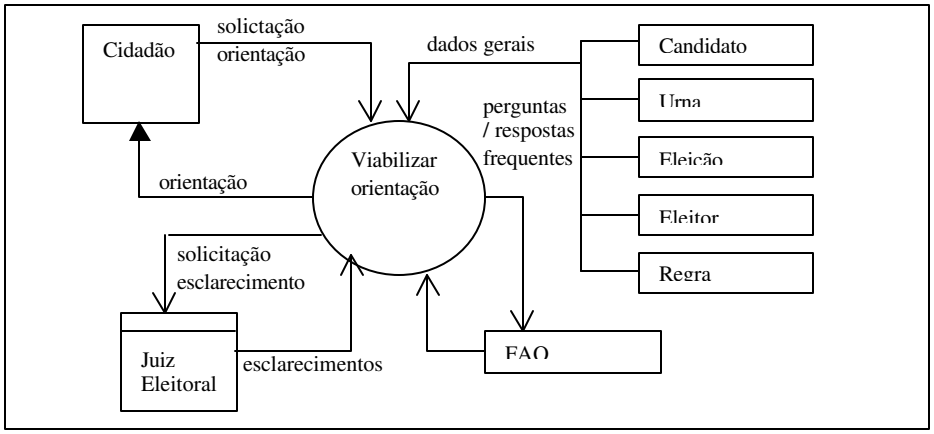
EC - V03 - DRE - 02. Eleitor solicita permissão para votar



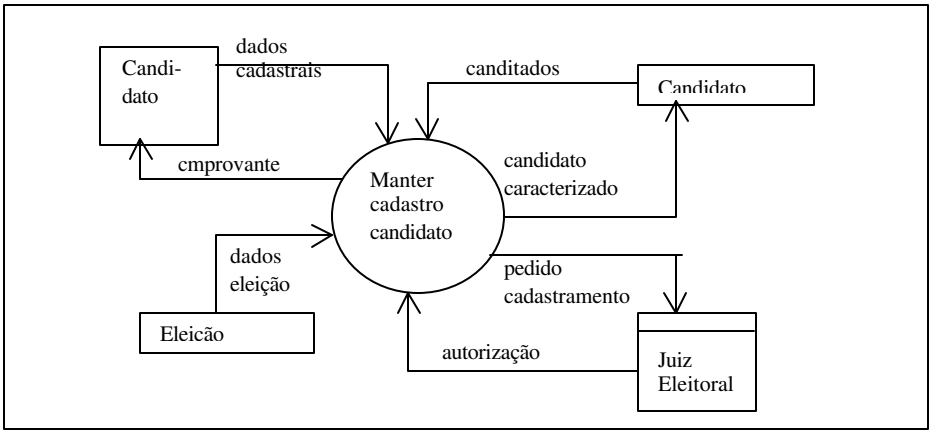
EC - V03 - DRE - 03. Eleitor realiza voto



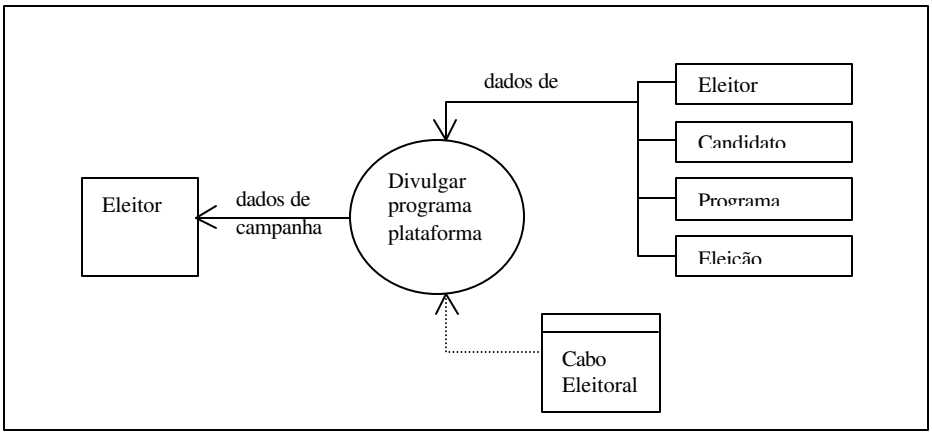
EC - V03 - DRE – 04, 06, 09 "Cidadão" solicita orientação



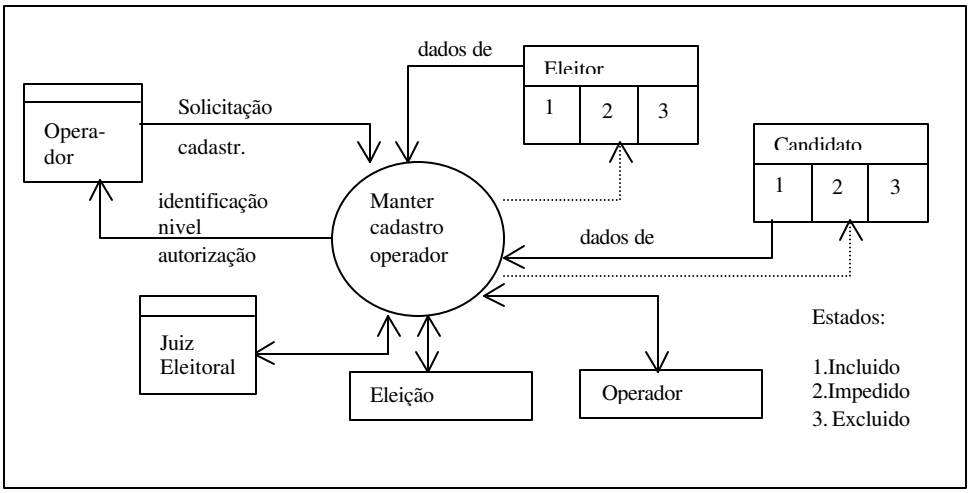
EC - V03 - DRE - 05. Candidato solicita cadastramento



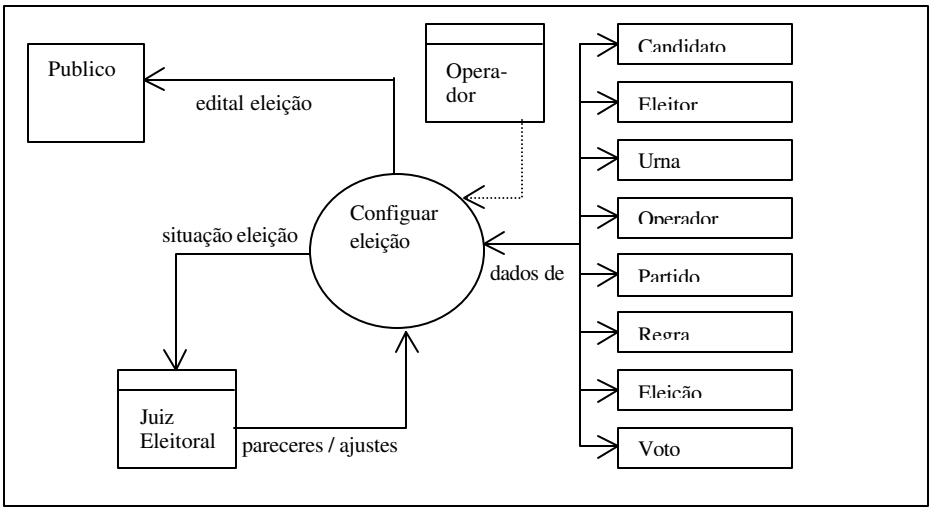
EC - V03 - DRE - 07. Momento de divulgar programa / plataforma



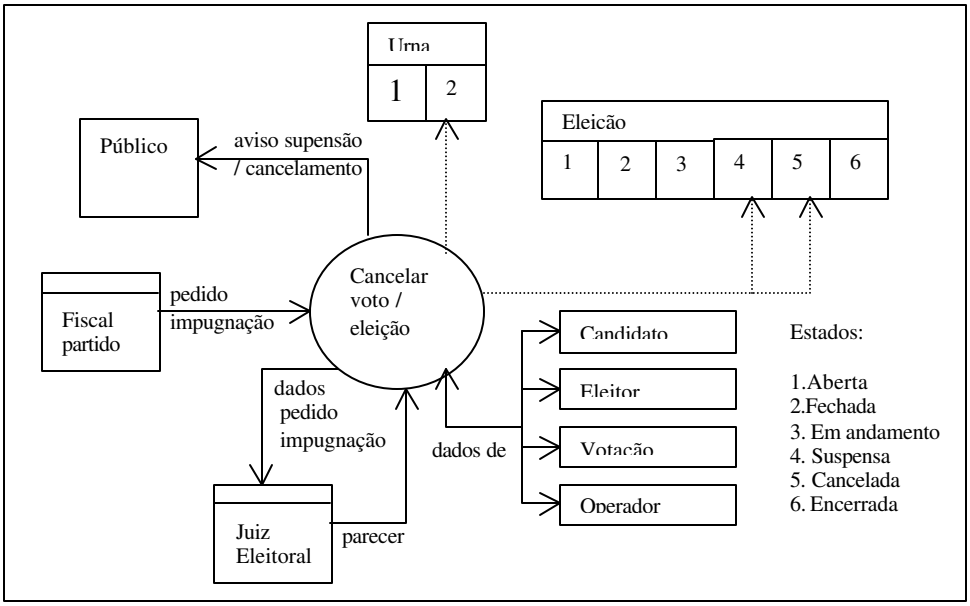
EC - V03 - DRE - 08. Operador solicita cadastramento



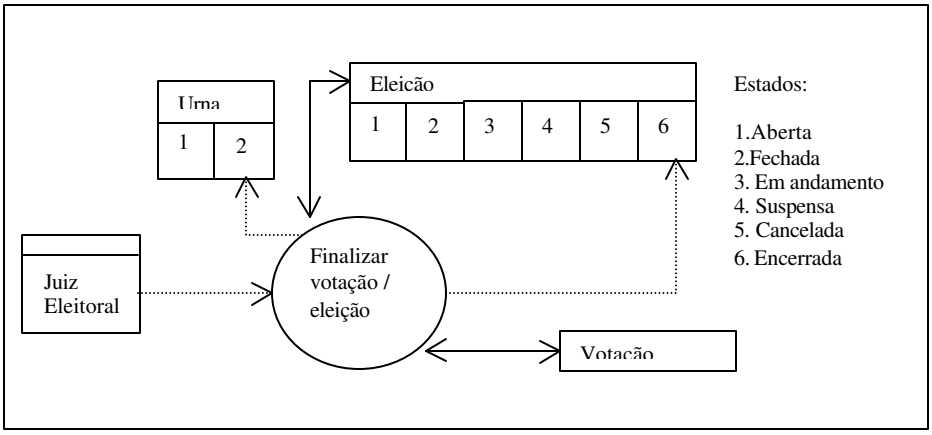
EC - V03 - DRE - 10. Momento de configurar eleição



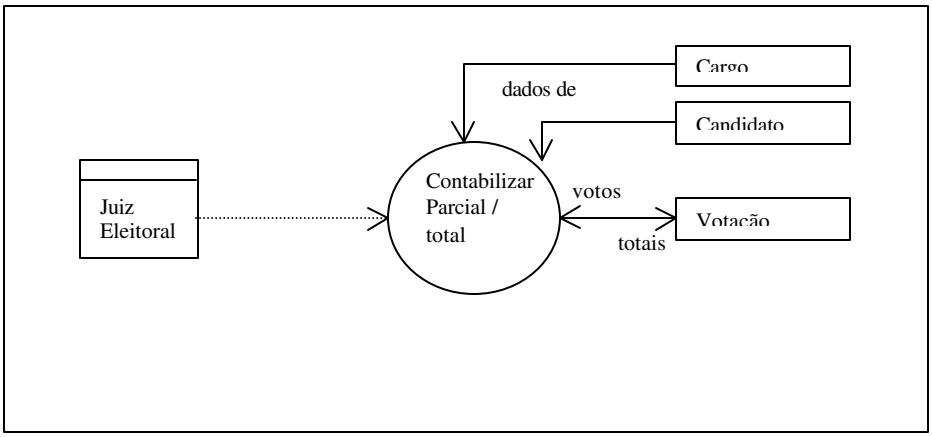
EC - V03 - DRE - 11, 12, 13, 14. Fiscal / Juiz "mela" / cancela voto / eleição



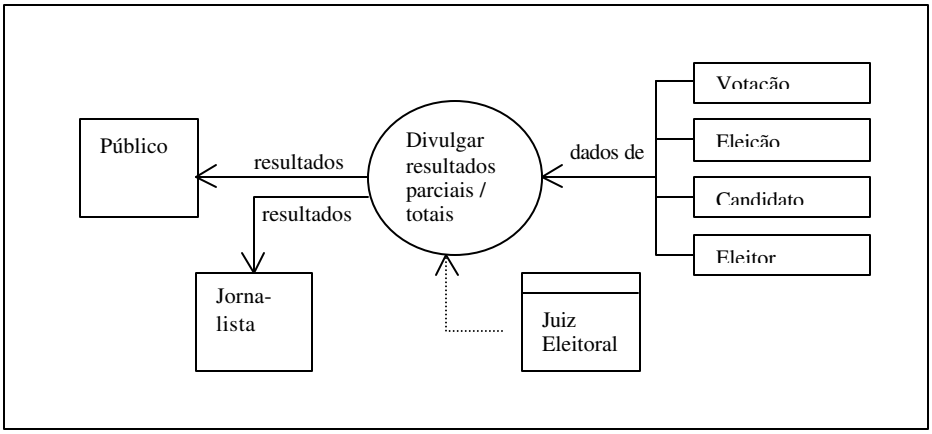
EC - V03 - DRE – 15, 16. Momento de finalizar votação / eleição



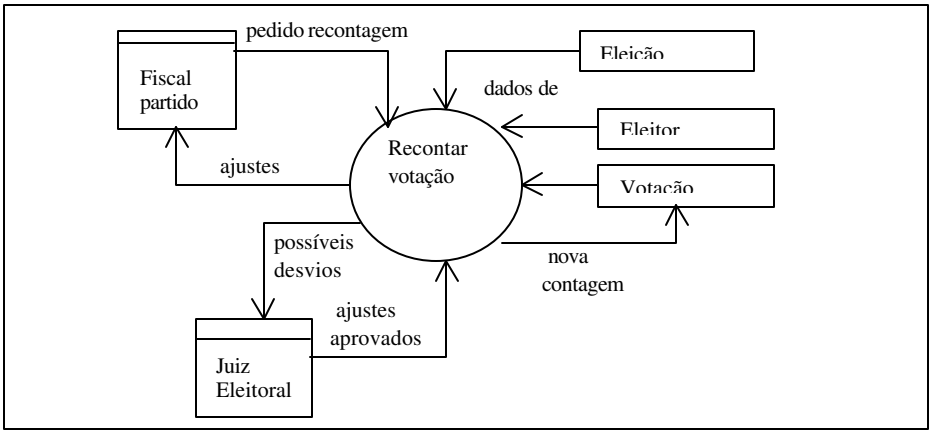
EC - V03 - DRE - 17. Momento de apurar os votos



EC - V03 - DRE - 18. Momento de divulgar resultados



EC - V03 - DRE - 19. Fiscal partido solicita recontagem



.VISÃO 04

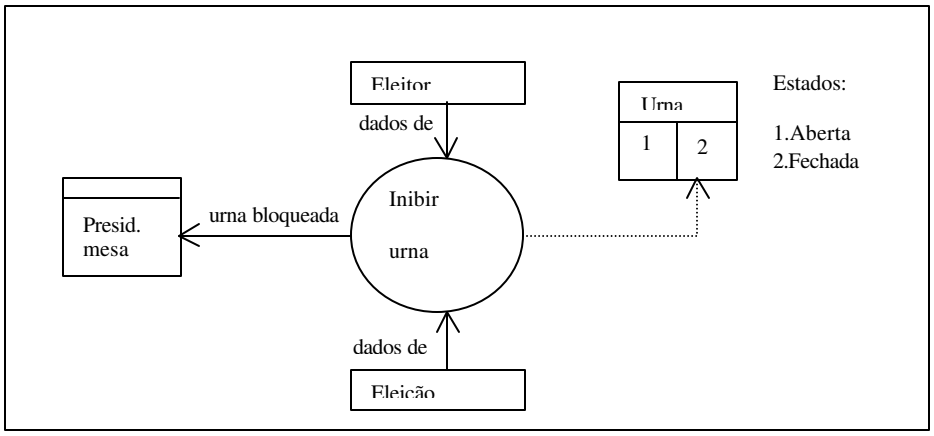
Esta visão foi diferenciada da visão de referência 03 no valor de *Tipo-Plano* para *produção*. A necessidade tem característica relativa ao nível de serviço operacional.

Referência-Visão : V04
Referência-Limite : L1
Referência-Foco : F2
Referência-Plano : P2

Referência-Foco : F2
Tipo-Foco : Função
Abordagem : Sistema
Estratégia : *bottom-up*
Tipo-representação : Gráfica
Representação : DRE - Reação a Eventos - 20

Referência-Plano : P2
Tipo-Necessidade : Requisito
Natureza-Necessidade: Funcional
Tipo-Plano : Produção

EC - V04 - DRE - 20. Momento de inibir urna por *time-out*



.VISÃO 05

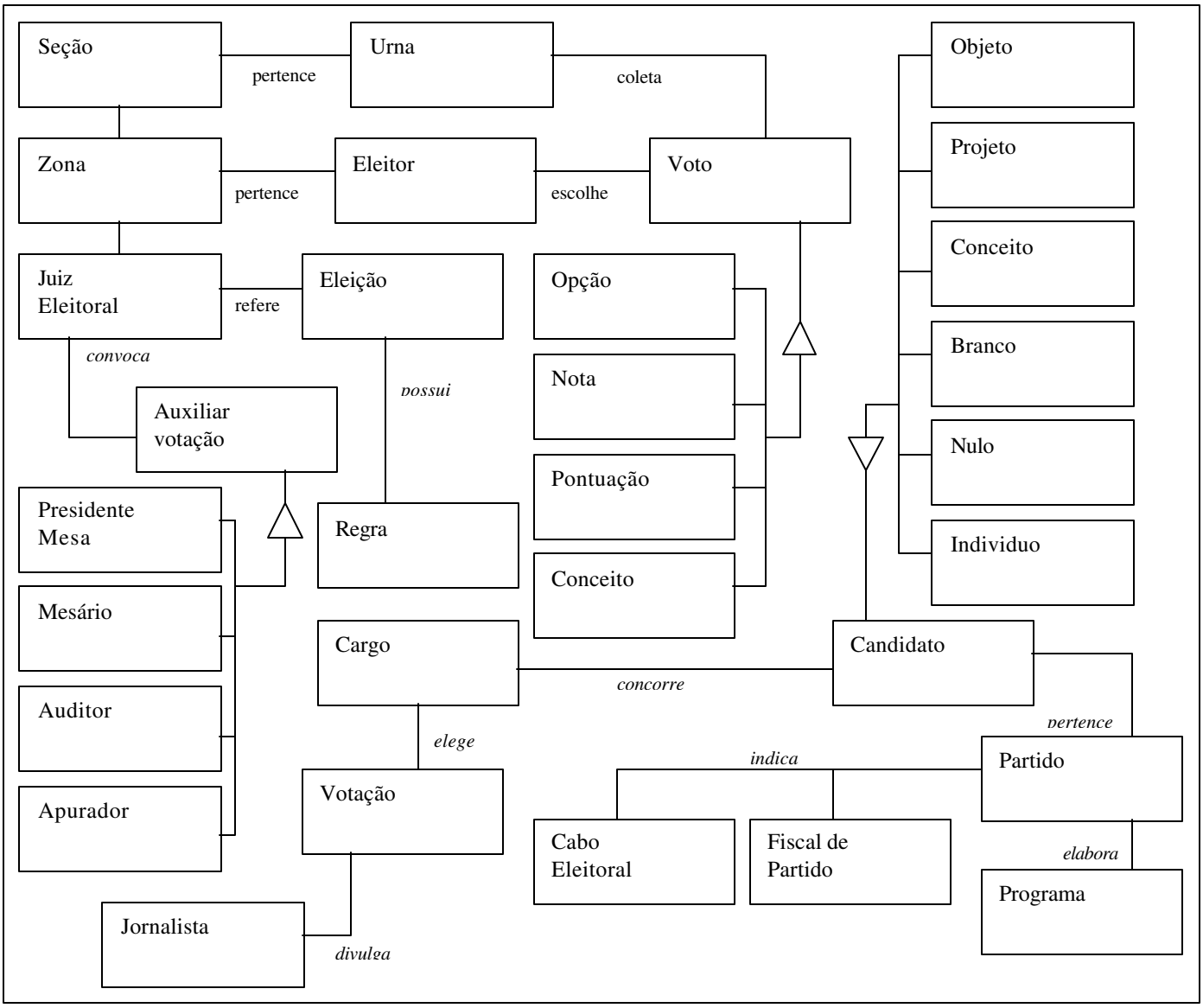
Esta visão foi determinada pela abordagem *OO* . Destaca-se que o adjetivo *genérica* é tratado por meio dos vários tipos especializados de *votos* e de *candidatos*. As possíveis associações entre as classes não se esgotam no modelo apresentado.

Referência-Visão : V05
Referência-Limite : L0
Referência-Foco : F1
Referência-Plano : P1
Observação : -----

Referência-Foco : F1
Tipo-Foco : Dado
Abordagem : OO
Estratégia : *bottom-up*
Observação : -----
Tipo-representação : Gráfica
Representação : DAC - Diagrama de Associação de Classes - Eleição Genérica (p1)

Referência-Plano : P1
Tipo-Necessidade : Requisito
Natureza-Necessidade: Funcional
Tipo-Plano : Utilização

EC - V05 - DAC - Eleição Genérica (p1)



.VISÃO 06

Esta visão foi determinada levantando-se os atributos estáticos e dinâmicos de cada classe de objetos. As visões de referências 14 e 15 (elementos e estrutura de dados) podem ser, no caso, elaboradas em paralelo às descobertas dos atributos estáticos (estrutura). A visão de referência 09 (módulos de software) , visão de referência 10 (diagrama de blocos) e outra qualquer relativa à especificação de funções com representação na forma textual (*DNL*) podem ser elaboradas, no caso, em paralelo às descobertas dos atributos dinâmicos (comportamento).

Referência-Visão : V06
Referência-Limite : L0
Referência-Foco : F3
Referência-Plano : P1
Observação : -----

Referência-Foco : F3
Tipo-Foco : Comportamento
Abordagem : OO
Estratégia : *bottom-up*
Observação : -----
Tipo-representação : Gráfica
Representação : DCL - Diagrama de Classes - Eleição Genérica

Referência-Plano : P1
Tipo-Necessidade : Requisito
Natureza-Necessidade: Funcional
Tipo-Plano : Utilização

EC - V06 - DCL - Eleição Genérica

Eleitor	Voto	Candidato	Partido
Identificação [Nome] [Nome-mãe] [Data-nascimento] [Identificação-Externa] Contato Local	Referência Tipo-voto [Comentário]	Identificação Tipo-candidato [Nome] [Descrição] [Currículo] [Imagem] [Identificação-Externa] [Contato] [Identificação-Eleitor] [Atributo]* [Peso]*	Número [Sigla] Nome [Símbolo] [Lema] [Cor]
Votar		Divulgar programa	

Zona	Seção	Urna	Cargo
Número [Nome] [Descricão] [Local]	Número [Nome] [Descricão] [Local]	Referência [Local] Tipo-interface	Referência [Tipo] [Descrição] Qtde-vagas
		Exibir candidato Exibir opções Coletar Voto	

Eleição	Votação	Programa	Apurador
Referência [Descrição] [Turno] [Data/Hora-início] [Data/Hora-fim] [Qtde-voto-por-eleitor]	Referência {Data-votação] [Tipo-contagem] [Tipo-resultado] Valor-Contagem* [Qtde-voto-efetuado]	[Resumo] [Descrição] [Item-destaque]*	Identificação [Identificação-Externa] Tipo-Cálculo
	Obter resultados		Contabilizar Voto

EC – V06 - DCL - Eleição Genérica

Regra	Jornalista	Juiz Eleitoral	Presidente mesa
Código-Regra Descrição Tipo-Cálculo-contagem Cálculo-contagem*	Nome-instituição	Identificação [Nome] [Identificação-Externa]	Identificação [Nome] [Identificação-Externa]
	Divulgar resultados	Iniciar eleição Cancelar eleição Cancelar voto Encerrar eleição Julgar desvios Convocar omissos Convocar auxiliares	Liberar urna Inibir urna

Cabo eleitoral	Auditor	Fiscal partido	Mesário
Identificação [Identificação-Externa] [Nome]	Identificação [Identificação-Externa] [Nome]	Identificação [Identificação-Externa] [Nome]	Identificação [Identificação-Externa] [Nome]
Promover candidato Divulgar programa	Conferir regras Conferir contagem Apresentar desvios	Fiscalizar regras Cancelar voto Cancelar eleição	

Edital	Auxiliar-votação		
Referência Data-emissão Referência-eleição [Texto]	Identificação [Nome] [Identificação-Externa] Contato		
Convocar eleitores	Orientar Eleitor Orientar Candidato		

.VISÃO 07

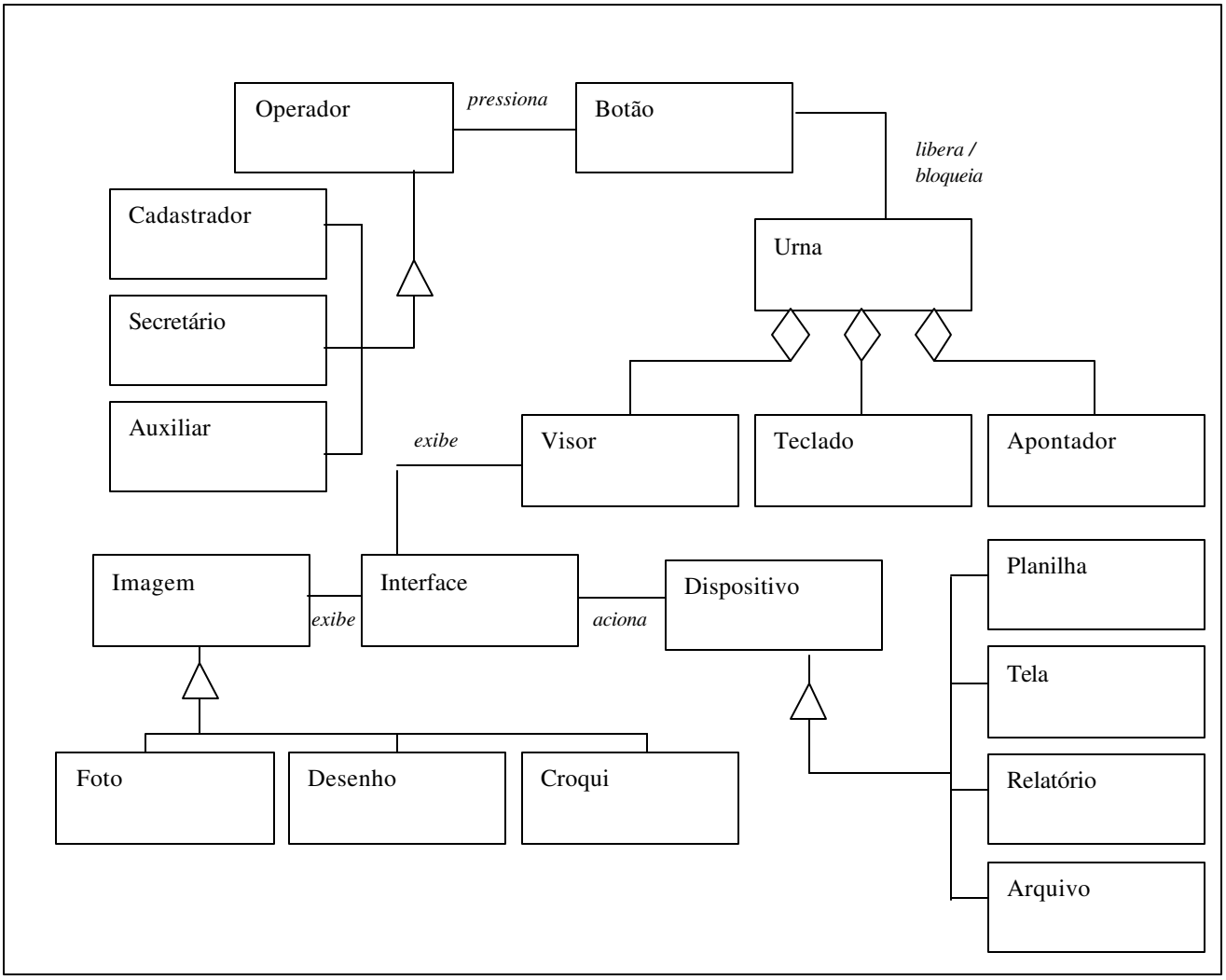
Esta visão é semelhante à visão de referência 06 com classes de objetos pertencentes ao contexto da tecnologia, ao *Tipo-Plan*o de valor *software-construção*. É dado tratamento separado pois essas classes têm maior potencial de reuso. Esta visão poderá ser utilizada em outros projetos.

Referência-Visão : V07
Referência-Limite : L0
Referência-Foco : F1
Referência-Plano : P4
Observação : -----

Referência-Foco : F1
Tipo-Foco : Dado
Abordagem : OO
Estratégia : *bottom-up*
Observação : -----
Tipo-representação : Gráfica
Representação : DAC - Diagrama de Associação Classes - Eleição Genérica (p4)

Referência-Plano : P4
Tipo-Necessidade : Requisito
Natureza-Necessidade: Funcional
Tipo-Plano : Software - Construção

EC - V07 - DAC - Eleição Genérica (p4)



.VISÃO 08

Como na visão 06, esta visão foi determinada levantando-se os atributos estáticos e dinâmicos de cada classe de objetos. As visões de referências 14 e 15 (elementos e estrutura de dados) podem ser, no caso, elaboradas em paralelo às descobertas dos atributos estáticos (estrutura). A visão de referência 09 (módulos de software) , visão de referência 10 (diagrama de blocos) e outra qualquer relativa à especificação de funções com representação na forma textual (*DNL*) podem ser elaboradas, no caso, em paralelo às descobertas dos atributos dinâmicos (comportamento).

Referência-Visão : V08
Referência-Limite : L0
Referência-Foco : F2
Referência-Plano : P4
Observação : -----

Referência-Foco : F2
Tipo-Foco : Função
Abordagem : OO
Estratégia : *bottom-up*
Observação : -----
Tipo-representação : Gráfica
Representação : DCL – Diagrama de Classes - Eleição Genérica

Referência-Plano : P4
Tipo-Necessidade : Requisito
Natureza-Necessidade: Funcional
Tipo-Plano : Software - construção

EC - V08 - DCL - Eleição Genérica

Arquivo	Operador	Urna	Teclado-urna
Referência Nome Tipo-arquivo [Descrição] [Retenção] [Data-geração] [Versão]	Identificação [Tipo-operador] [Descrição] Nível-autorização [Nome-papel] Contato	Referência Valor-*time-out*	Tecla Significado
Armazenar-dados	Disponibilizar aplicação Cadastrar-dados Atribuir-papel Atrib.-nivel-autorização	Inibir (por *time-out*)	Receber voto

Apontador-urna	Visor-urna	Planilha - papel	Planilha - tela
Tipo-apontador Aparência		Referência [Título] Referência-eleição [Data-emissão] [Página] [Versão]	Referência [Título] Referência-eleição
Selecionar candidato Selecionar botão	Exibir candidato Exibir voto Exibir botões	Coletar-dados	Coletar-entradas Validar-entradas

Relatório-papel	Relatório-tela	e-1/2	Botão
Referência Data-emissão Página [Versão] [Título] Referência-eleição	Referência [Título] Referência-eleição	Identificação-emissor Identificação-receptor Mensagem	Referência Efeito Significado-opção Aparência
Exibir dados		Transportar-dados	Direcionar opção

.VISÃO 09

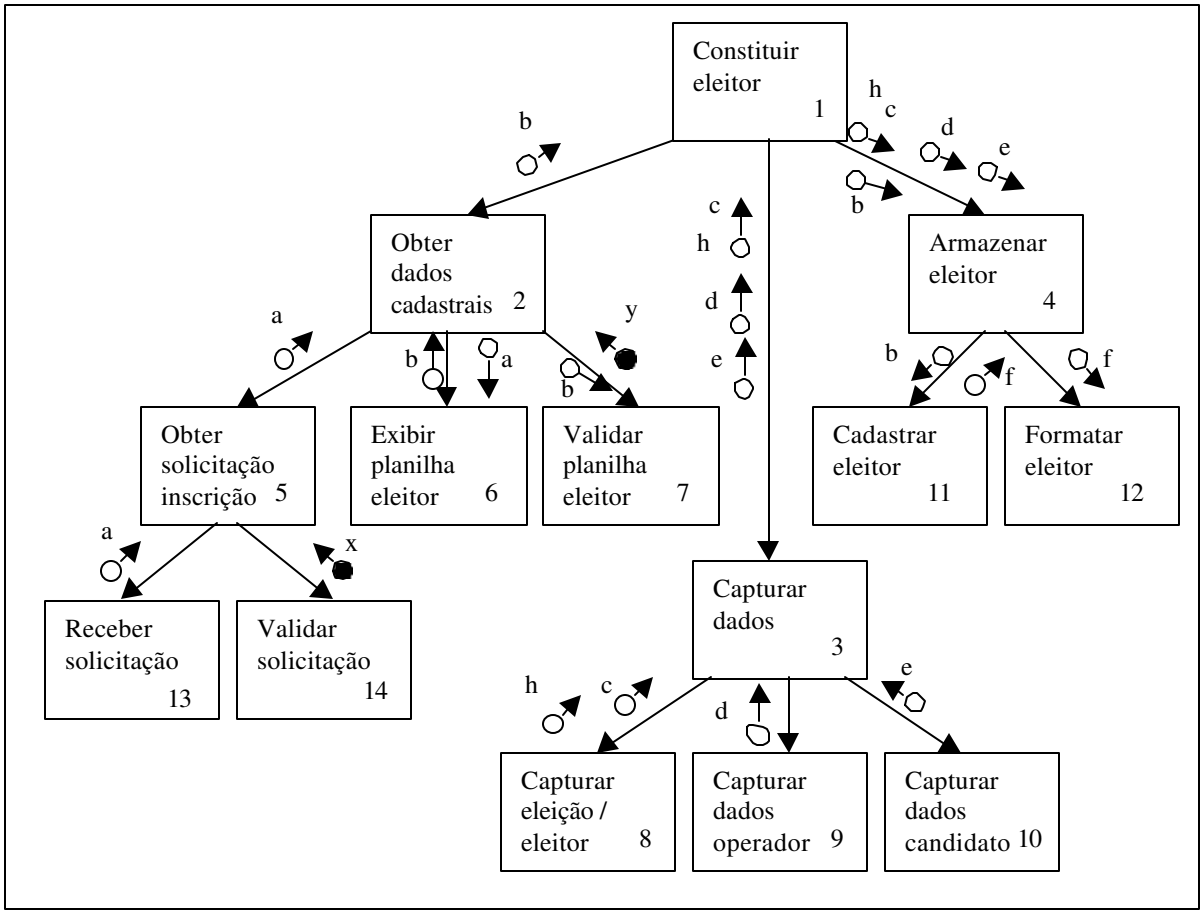
Esta visão foi determinada, a partir da visão de referência 03. Utilizou-se o esquema proposto no item 3.7 *Análise de Componentes*. Foram agregados os componentes *administrativos*, de *controle de padrão* e de *transporte* para a operação essencial *cadastrar eleitor*. Foi também utilizada a Tabela 3.2 de verbos sugeridos para nomear os componentes.

Referência-Visão : V09
Referência-Limite : L0
Referência-Foco : F2
Referência-Plano : P4
Observação : -----

Referência-Foco : F2
Tipo-Foco : Função
Abordagem : OO
Estratégia : *bottom-up*
Observação : -----
Tipo-representação : Gráfica
Representação : DES - Diagrama de Estrutura - Constituir Eleitor

Referência-Plano : P4
Tipo-Necessidade : Requisito
Natureza-Necessidade: Funcional
Tipo-Plano : Software-construção

EC - V09 - DES - Constituir Eleitor



Legenda:

a - solicitação de cadastramento
b - dados cadastrais eleitor
c - última identificação
d - dados de operador existente
e - dados de candidato existente
f - identificação eleitor
h - dados eleição
x - ok / ñok
y - ok / ñok

Tipo de Componentes:

Administrativo : 1, 2, 3, 4, 5
Controle padrão : 7, 12, 14
Transporte : 6, 8, 9, 10, 13
Essencial : 11

.VISÃO 10

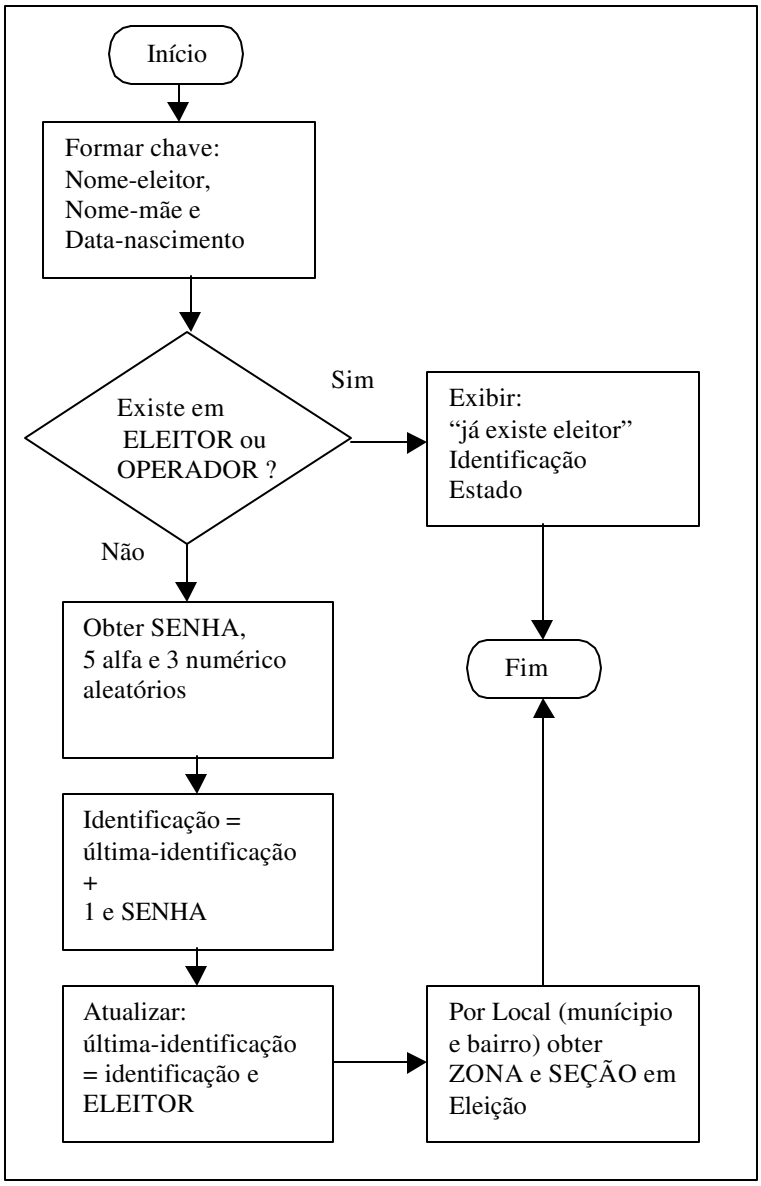
Esta visão consiste na especificação, através de uma representação gráfica, o diagrama de blocos, para o componente essencial *cadastrar eleitor* da visão de referência 09.

Referência-Visão : V10
Referência-Limite : L0
Referência-Foco : F2
Referência-Plano : P4
Observação : -----

Referência-Foco : F2
Tipo-Foco : Função
Abordagem : OO
Estratégia : *bottom-up*
Observação : -----
Tipo-representação : Gráfica
Representação : DBL - Diagrama de Bloco - Cadastrar Eleitor

Referência-Plano : P4
Tipo-Necessidade : Requisito
Natureza-Necessidade: Funcional
Tipo-Plano : Software - construção

EC - V10 - DBL - Cadastrar Eleitor



.VISÃO 11

Esta visão foi gerada a partir dos atributos estáticos (estrutura) das classes de objetos, no caso a visão de referência 06 e de outras visões que modelem os requisitos/restrições de *Tipo-Plano* com valor *produção*. Conforme proposto esses requisitos/restrições podem ter sido obtidos a partir de dispositivos físicos clássicos (no caso as planilhas em papel). Aqui estão exemplificados apenas os conteúdos de duas planilhas. Pode-se ter uma outra visão, elaborada a partir desta, que represente o *lay-out* de cada uma dessas planilhas.

Referência-Visão : V11
Referência-Limite : L0
Referência-Foco : F1
Referência-Plano : P4
Observação : -----

Referência-Foco : F1
Tipo-Foco : Dado
Abordagem : OO
Estratégia : *bottom-up*
Observação : -----
Tipo-representação : Textual
Representação : Planilhas - Papel

Referência-Plano : P4
Tipo-Necessidade : Requisito
Natureza-Necessidade: Funcional
Tipo-Plano : Software - construção

EC - V11 - Planilhas - papel

1. Inscrição Eleitor

EC - P1 - INSCRIÇÃO ELEITOR - V1.0 Página única

Nome-Eleitor
Nome-mãe
Data-nascimento
Data-preenchimento
Identificação-externa
    Tipo {RG, CPF, Título,outra}
    Valor
Contato
    Logradouro
    Número
    [Complemento]
    Município
    [CEP]
    [Telefone]*
    [e-1/2]*

2. Inscrição Candidato

EC - P2 - INSCRIÇÃO CANDIDATO - V1.0 Página única

Tipo-candidato {indivíduo, projeto, objeto, conceito, outro}
[Identificação-eleitor]
Data-preenchimento
[Identificação-externa]
    Tipo {RG, CPF, Título, outra}
    Valor
[Contato]
[Currículo]
[Imagem]
[Atributo]*
[Peso]
[Descrição]*

.VISÃO 12

Esta visão foi elaborada a partir do ambiente de produção desejado para implantação operacional da aplicação de software em questão. O principal fator para esta escolha foi a predominância de tal tecnologia no mercado.

Referência-Visão : V12
Referência-Limite : L0
Referência-Foco : F1
Referência-Plano : P2
Observação : -----

Referência-Foco : F1
Tipo-Foco : Dado
Abordagem : Sistema
Estratégia : *bottom-up*
Observação : -----
Tipo-representação : Textual
Representação : Relação tecnologia prevista

Referência-Plano : P2
Tipo-Necessidade : Restrição
Natureza-Necessidade: Não Funcional
Tipo-Plano : Produção

EC - V12 – Relação Tecnologia prevista

Sistema operacional

.MS Window 98 / 2000 / ME / XP

.UNIX

Linguagem programação

.JAVA (SUN Microsystems)

Data Base

.MS Access 2000

.ORACLE

Páginas Web

.DreamWeaver (Macromedia)

Plataforma / Ambiente

.J2EE – JAVA 2 Enterprise Edition (SUN Microsystems)

Equipe de desenvolvimento prevista:

. um Analista de Sistemas Sr. (análise de requisitos) - 160 horas
. um Projetista Sr. (conhecer ambiente J2EE) - 160 horas
. um Programador Java Sr. - 160 horas
. um Analista de Banco de Dados Pleno (Oracle) - 160 horas
. um Web Designer - 080 horas

Custo aproximado previsto em pessoal: salários US$7.000,00 ; encargos: US$5.000,00

.VISÃO 13

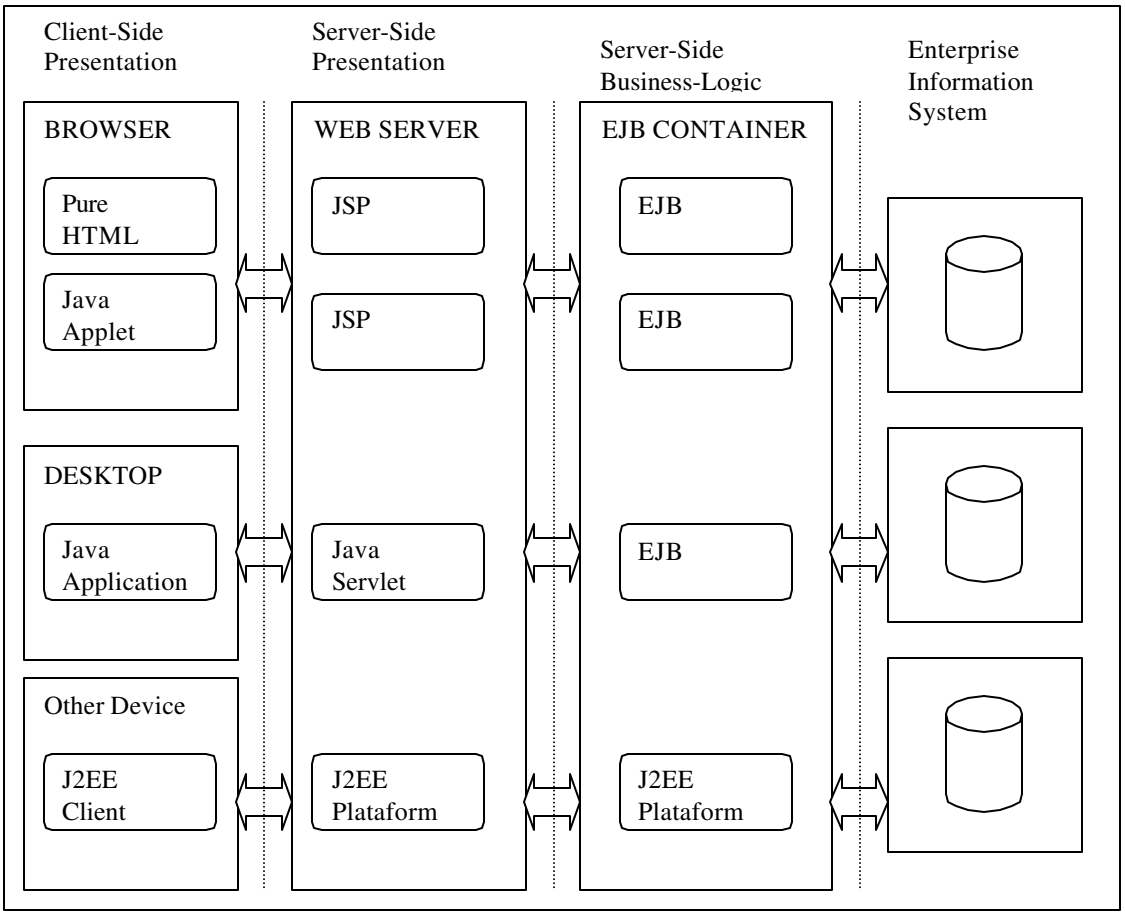
Esta visão foi gerada a partir de informações disponibilizadas pelo fornecedor do ambiente de desenvolvimento/produção previsto. A representação gráfica foi utilizada a partir de um esquema apresentado também pelo fornecedor de tal tecnologia. Apresenta de forma geral a tecnologia a ser adotada para implementação e operacionalização da aplicação de software que está sendo desenvolvida.

Referência-Visão : V13
Referência-Limite : L0
Referência-Foco : F1
Referência-Plano : P2
Observação : -----

Referência-Foco : F1
Tipo-Foco : Dado
Abordagem : Sistema
Estratégia : *botom-up*
Observação : -----
Tipo-representação : Gráfica
Representação : Diagrama Plataforma J2EE

Referência-Plano : P2
Tipo-Necessidade : Restrição
Natureza-Necessidade: Não Funcional
Tipo-Plano : Produção

EC - V13 - Diagrama - Plataforma ambiente J2EE



Legenda:

J2EE - JAVA 2 Enterprise Edition
JDBC - Java DataBase Conectivity (acesso aos DBs)
EJB - Enterprise JavaBeans (encapsula a lógica)
JSP - JAVA Server Page (interação com cliente)

.VISÃO 14

Esta visão foi gerada a partir de informações que surgiram na modelagem de outras visões. Elas devem ser implementadas através de uma ferramenta do tipo *Dicionário de Dados*. Os elementos de dados devem ser especificados durante o processo de desenvolvimento da aplicação de software. Essa visão possui alto potencial de reuso podendo ser utilizada em outros projetos.

Referência-Visão : V14
Referência-Limite : L0
Referência-Foco : F1
Referência-Plano : P1
Observação : -----

Referência-Foco : F1
Tipo-Foco : Dado
Abordagem : OO
Estratégia : *bottom-up*
Observação : -----
Tipo-representação : Textual
Representação : DD – Elemento de Dados – especificação tipo *atribuição.*

Referência-Plano : P1
Tipo-Necessidade : Requisito
Natureza-Necessidade: Funcional
Tipo-Plano : Utilização

EC - V14 – DD - Elemento de Dados

Nome : Identificação-Eleitor
Alias : ------
Descrição : Número que identifica unicamente cada eleitor
Classe de valor : discreta
Valor : 7 dígitos seqüenciais mais dois DV
Tamanho : 9 numéricos
Validade : numéricos e dois últimos algarismos fechando Módulo 11.
Observação : Obtido pela chave nome/nome-mãe/data nascimento. Homônimos resolvidos *ad hoc*.

Nome : Tipo-Identificação-Externa
Alias : ------
Descrição : Número que identifica tipo documento.
Classe de valor : discreta
Valor : 1. RG, 2. CPF, 3. Título Eleitor, 4. Carteira motorista, 5. outro.
Tamanho : 1 numérico.
Validade : numérico de 1 a 5.
Observação : ------

Nome : Tipo-Voto
Alias : ------
Descrição : ------
Classe de valor : discreta
Valor : 1. Opção, 2. Nota, 3. Pontuação, 4. Conceito.
Tamanho : 1 numérico.
Validade : numérico de 1 a 4.
Observação : ------

Nome : Tipo-Resultado
Alias : ------
Descrição : ------
Classe de valor : discreta
Valor : 1.Parcial, 2. Total.
Tamanho : 1 numérico.
Validade : numérico de 1 a 2.
Observação : ------

EC - V14 - DD - Elemento de Dados

Nome : Tipo-Cargo
Alias : ------
Descrição : ------
Classe de valor : discreta
Valor : Vereador, Prefeito, Deputado Estadual, Deputado Federal, Senador, Governador, Presidente, Miss do Vinho, Miss Brasil, Miss Universo, Projeto Vencedor, Homem do ano, etc.
Tamanho : até 30.
Validade : alfabético e constante na tabela de valor..
Observação : ------

Nome : Tipo-Cálculo
Alias : ------
Descrição : ------
Classe de valor : discreta
Valor : 1. Contagem, 2. Média aritmética, 3. Média ponderada.
Tamanho : 1 numérico.
Validade : numérico de 1 a 3.
Observação :-----

.VISÃO 15

Esta visão foi gerada a partir de informações que surgiram na modelagem de outras visões. Elas devem ser implementadas através de uma ferramenta do tipo *Dicionário de Dados*. As estruturas de dados devem ser especificadas durante o processo de desenvolvimento da aplicação de software. Essa visão possui alto potencial de reuso podendo ser utilizada em outros projetos.

Referência-Visão : V15
Referência-Limite : L0
Referência-Foco : F1
Referência-Plano : P1
Observação : -----

Referência-Foco : F1
Tipo-Foco : Dado
Abordagem : OO
Estratégia : *bottom-up*
Observação : -----
Tipo-representação : Textual
Representação : DD – Estrutura de dados – especificação tipo *atribuição*.

Referência-Plano : P4
Tipo-Necessidade : Requisito
Natureza-Necessidade: Funcional
Tipo-Plano : Utilização

Nome : Identificação-Externa
Descrição : -------
Conteúdo :
Tipo-identificação
Valor

Observação : ------

Nome : Contato
Descrição : -------
Conteúdo
Logradouro
Número
[Complemento]
Município
[CEP]
[Telefone]*
[e-1/2]*

. Comentários e considerações

O meta modelo de referência da *Análise por Visões* permite o preenchimento da *Tabela de Necessidades* a partir do modelo descritivo. Mas não há a necessidade de se prender, em forma e /ou conteúdo, ao modelo descritivo. Ele representa apenas o ponto de partida. Colocam-se as idéias ou informações novas que possam surgir na execução da tarefa. O modelo de referência pode também auxiliar na organização das atividades de elicitação de requisitos.

Os requisitos podem ser agrupados por suas diversas características. Pode ser, por exemplo por valor de *Tipo-Plano*.

Os modelos obtidos são os mesmos de qualquer metodologia. Muda-se apenas o caminho prático, já que a proposta é uma técnica, para obtê-los. Não se prende a nenhum tipo de diagrama específico pertencente a um método ou outro. Qualquer ferramenta de representação pode ser utilizada segundo conhecimento e conveniência do modelador. Como as necessidades já estão caracterizadas e melhor segmentadas, ou seja, os escopos já estão menores, consegue-se maior facilidade de modelagem e melhor qualidade nos modelos gerados. Observa-se que pode haver divergências de características de cada *Visão*, pois sendo uma técnica prática de modelagem, os modelos sairão segundo a visão de cada modelador.

A modelagem dos *Limites* já particionado, deixam os modelos mais claros, mais simples e mais legíveis. Um modelo único, como recomendado pelos métodos estruturados, um único *Diagrama de Contexto*, seria, pelo excesso de informações, maior, mais poluído, esteticamente mais feio. Seria também mais difícil de idealizar e desenhar.

Nos modelos particionados não houve a preocupação de integração através das *Entidades Externas,* como recomendam os métodos estruturados. Por exemplo, a macrofumção *Preparar Eleição* sendo uma *Entidade Ext*erna para a macrofunção *Realização Votação* e assim por diante. A integração pode ser melhor modelada através de

uma outra *visão*. Essa outra visão pode ser, por exemplo, com *Tipo-Foco* com valor *dado*, *abordagem* com valor *Sistema* e *Tipo-Plano* com valor *utilização*.

O *Limite* tratado como um todo, visão de referência 00, foi útil para a elaboração do diagrama de *Use-Cases*, visão de referência 01. A partir dos *DLSs* identificaram-se os atores e as funcionalidades visíveis a eles. Não foram efetuadas as *explosões* dos *DFDs* dos *Limites* como sugerem os métodos estruturados. Após elaboração do diagrama de *Use-Cases,* foi abstraída a *Relação de Eventos*, visão de referência 02. Não foram elaborados modelos descritivos de cada *Use-Case* como recomendado pela OOSE.

Vários métodos e ferramentas de representação foram utilizados de metodologias diferentes, sem problemas. No caso descrito foram combinadas a *Análise Estruturada* (DFD) [Gane79], *OOSE* (*Use-Cases*) [Jacobson92] e *Análise Essencial* (Relação de Eventos) [McMenamim84].

Destaca-se que não está sendo proposta uma seqüência de passos com as respectivas ferramentas de representação a serem adotadas. Não está sendo proposta uma metodologia e nem um novo modelo de ciclo-de-vida [Yourdon92]. Esta seqüência foi interessante para este estudo de caso. Pode ser que não seja para outro tipo de aplicação de software ou outra circunstância de desenvolvimento. Assim, neste sentido, esta idéia aproxima-se bastante do proposto pelo *RUP* [Jacobson01], onde cada projeto de desenvolvimento de uma aplicação de software é uma *instância* do processo geral. Frente à diversidade apenas a diversidade [Ashby70].

A elaboração da lista de eventos, visão de referência 02, foi extraída a partir tanto dos *DLSs* parciais dos *Limites* como do diagrama geral de *Use-Cases*.

O fato de caracterizar os eventos segundo tabela e responder às 8 perguntas, como sugere a *Análise de Eventos*, auxilia em muito, não só a futura modelagem das reações dos eventos, a confecção dos *DREs* visão de referência 04, como já melhora a compreensão de cada evento. Os agentes externos, neste estudo de caso, são apenas os atores que

representam os papéis de *Eleitor* e *Candidato.* Os demais agentes, por exemplo o *Juiz Eleitoral* e o *Presidente de Mesa*, representam papéis *ad hoc* do próprio sistema *Eleição Genérica*.

O *Plano* que , categorizando necessidades, produziu visões separadas de classes de objetos, visões de referências 06 e 08, pode auxiliar e facilitar a *Análise de Domínios* [Pressman97].

A visão de referência 09, o *Diagrama de Estrutura*, mostra que ele é adequado, quanto a fatores estéticos e facilidade de elaboração, quanto menor for o contexto. Foi categorizado como abordagem *OO*, para orientar o modelador a usá-lo em nível de *operação* de classe de objeto. A mesma orientação vale para o *DRE - Diagrama de Reação a Eventos*, elaborado na visão de referência 03. Para este estudo de caso os verbos *constituir*, *armazenar* e *formatar* devem ser substituídos para uma melhor clareza no modelo *Cadastrar Eleitor*.

A classe *Candidato* é fundamental, neste estudo de caso, para possibilitar a flexibilidade, para possibilitar o *genérica* da eleição. Percebe-se isto pelo quantidade de propriedades opcionais da classe *Candidato* na visão de referência 06. Mesmo pulverizando estas propriedades em classes especializadas, esta quantidade não seria minimizada. Notar que se o *Candidato* for do tipo *Objeto*, por exemplo *Escola de Samba*, (quesitos: formação, bateria, conjunto, etc.) ou *Indivíduo, Concurso de Beleza,* por exemplo, (quesitos: cintura, busto, etc.) teriam ambos os tipos a possibilidade de possuírem atributos com *peso* associado. Um de seus atributos, *Imagem*, foi especializado em outra visão, a de referência 07.

Pode-se incluir na documentação um diagrama não padrão, didático, apenas ilustrativo, declarando sua abstração por meio da determinação de sua visão. Normalmente em fase de projeto surgem muitos desses tipos de esquemas e diagramas. A visão determinada pode facilitar o entendimento dos mesmos. Tem-se como exemplo, neste estudo de caso, o *Diagrama plataforma J2EE*, visão de referência 13.