

PUC

Série: Monografias em Ciência da Computação,
No. 7/91

PROJETO ("DESIGN") NA ÁREA DE MÉTODOS ESTRUTURADOS:
TRANSFORMANDO REDES EM HIERARQUIAS

Anderson A. André
Bruno Maffeo

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP-22453
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA

Série: Monografias em Ciência da Computação, No. 7/91

Editor: Carlos J. P. Lucena

Maio, 1991

PROJETO ("DESIGN") NA ÁREA DE MÉTODOS ESTRUTURADOS:
TRANSFORMANDO REDES EM HIERARQUIAS *

Anderson A. André
Bruno Maffeo

* Trabalho patrocinado pela Secretaria de Ciência e Tecnologia da
Presidência da República.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio - Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ
Brasil

Tel.:(021)529-9386

Telex:31078

Fax:(021)511-5645

E-mail:rosane@inf.puc-rio.br

RESUMO: Este trabalho apresenta um estudo realizado na área Métodos Estruturados, voltado para projeto ("design") de sistemas. São apresentadas estratégias de transformação de um diagrama em rede para um diagrama de estrutura hierárquica, bem como os diversos critérios de avaliação e refinamento desse diagrama de estrutura. O estudo ressalta o interesse em evitar-se distorções ocorridas quando essa transformação é realizada.

PALAVRAS-CHAVES: Diagramas em Rede, Análise de Transação, Análise de Transformação, Módulo, Diagramas de Estrutura Hierárquica, Acoplamento, Coesão, Complexidade, Reusabilidade.

ABSTRACT: This work presents a study in the Structured Methods Area which aims at the design of systems. Strategies to transform a network schema to a hierarchical schema are presented, as well as many criteria to the structured chart refinement. A major concern during this study was to indicate the necessity of avoiding possible distortions when this transformation is made.

KEY-WORDS: Network Charts, Transaction Analysis, Transform Analysis, Module, Hierarchically Structured Charts, Coupling, Cohesion, Complexity, Reusability.

INTRODUÇÃO	1
1. TRANSFORMANDO UMA REDE QUE MODELA UM PROCESSO EM HIERARQUIAS	
1.1. PRINCÍPIOS BÁSICOS	2
1.1.1. O Módulo	2
1.1.2. Representação Gráfica de Um Módulo	3
1.1.3. Conexões Entre Módulos — Chamadas	3
1.1.4. A Comunicação Entre Módulos	4
1.1.5. A Área de Dados Compartilhados	4
1.1.6. A Especificação de Módulos	4
1.1.7. A Especificação de Dados	5
1.2. ANÁLISE DE TRANSAÇÃO	7
1.2.1. Centro de Transações em Sistemas "On-Line"	8
1.2.2. Identificação de Transações em Esquemas de Atividades	9
1.2.3. Observação Final	9
1.3. ANÁLISE DE TRANSFORMAÇÃO	10
1.3.1. Passo 1: Identificar o Centro de Transformação	11
1.3.2. Passo 2: Produzir Um Diagrama de Estrutura Preliminar	12
1.4. BIBLIOGRAFIA	14
2. REFINAMENTO DE UM DIAGRAMA DE ESTRUTURA	
2.1. MOTIVAÇÃO	15
2.2. ACOPLAMENTO	15
2.2.1. Tipos de Acoplamento	16
2.2.1.1. Acoplamento de Dados	16
2.2.1.2. Acoplamento de Imagem	16
2.2.1.3. Acoplamento de Controle	16
2.2.1.4. Acoplamento Comum	17
2.2.1.5. Acoplamento de Conteúdo	18

2.2.2. Obtendo Um Baixo Acoplamento	19
2.2.2.1. Criação de Estruturas de Dados	19
2.2.2.2. Utilização de Areas de Dados Compartilhados	20
2.2.2.3. Eliminação de Acoplamentos de Controle	20
2.2.3. Determinação do Tipo de Acoplamento	21
2.2.4. Observações Finais	21
2.3. COESÃO	23
2.3.1. Tipos de Coesão	24
2.3.1.1. Coesão Funcional	24
2.3.1.2. Coesão Seqüencial	24
2.3.1.3. Coesão Comunicacional	24
2.3.1.4. Coesão Procedimental	25
2.3.1.5. Coesão Temporal	26
2.3.1.6. Coesão Lógica	27
2.3.1.7. Coesão Coincidental	28
2.3.2. Determinação da Coesão de Um Módulo	29
2.3.3. Observações Finais	29
2.4. COMPLEXIDADE DE MÓDULOS	30
2.4.1. Fan-out	30
2.5. REUSABILIDADE DE MÓDULOS	32
2.5.1. Fan-in	32
2.6. BIBLIOGRAFIA	33
CONCLUSÃO	34
FIGURAS	

INTRODUÇÃO:

As ferramentas e técnicas de modelagem colocadas à disposição dos projetistas ("designers") de software pela Área de Métodos Estruturados dão, à primeira vista, a impressão de que a fase de projeto ("design") é simples. Uma visão pouco crítica conduziria a pensar que trata-se de processo quase automático obter um bom projeto ("design") a partir do Modelo da Essência. Entretanto, mesmo excluindo-se os aspectos relacionados com a vertente gerencial da Engenharia de Software, percebe-se que, na prática, não é bem assim.

Uma das principais dificuldades reside no fato de que o projeto ("design") de um software, quando são empregadas as ferramentas e técnicas de modelagem da Área de Métodos Estruturados, apresenta uma descontinuidade representacional.

Num dado momento, o projetista é obrigado a abandonar a "linguagem de redes" (DFDs, basicamente) e adotar uma "linguagem hierárquica", visando gerar o Diagrama de Módulos de um processo, programa etc., que emprega um princípio de subordinação baseado no protocolo de "chamada de sub-rotinas".

Não raramente, esse ponto de descontinuidade transforma-se num ponto de estrangulamento que dificulta a ação do projetista no sentido de obter um bom projeto ("design").

Entretanto, essa Área propõe duas estratégias, geralmente complementares, para superar essa dificuldade: a Análise de Transação e a Análise de Transformação. Este trabalho revisita a apresentação dessas duas estratégias, levando em consideração avanços conceituais mais recentes encontrados na literatura que trata desse tema [1].

Partindo de um bom projeto ("design"), as diretrizes de avaliação e refinamento propostas na Área de Métodos Estruturados podem conduzir o projetista a um excelente resultado final.

No entanto, se ele não aplicar adequadamente essas estratégias, o mais provável é a obtenção de um projeto ("design") medíocre, que obrigaria a realização de sucessivas modificações nos diagramas hierárquicos para obter alguma melhora do produto. Assim mesmo, essa melhora não está a priori assegurada e o processo de obtê-la é mais semelhante ao de uma "torcida para que dê certo" do que o resultado de ações bem programadas e racionais.

As seções que seguem apresentam um estudo das estratégias de projeto ("design") e das principais diretrizes de avaliação e refinamento existentes. Na seção 1, são apresentadas as estratégias designadas pelos termos "Análise de Transação" e "Análise de Transformação". Na seção 2, são apresentadas diretrizes que determinam a qualidade de um bom projeto.

1. TRANSFORMANDO UMA REDE QUE MODELA UM PROCESSO EM HIERARQUIAS

1.1. PRINCÍPIOS BÁSICOS

O mapeamento de um esquema de atividades primitivas que representam um processo em um diagrama de estrutura é a transformação de uma representação em rede em uma representação hierárquica. O resultado serve como base para se escrever o código.

Essa transformação deve introduzir o menor número possível de distorções. Para garantir isso, a transformação deve manter:

1º. a separação entre controle e operação;

2º. a parte de trabalho relacionada com o Modelo da Essência separada da parte relativa à implementação, onde prevalecem as características não-essenciais —tais como formatação de dados para operações de entrada e saída;

3º. as porções da hierarquia que movem dados separadas das porções que processam esses dados, buscando isolar porções de processamento independentes da especificidade da hierarquia, visando sua re-usabilidade.

As unidades básicas de um diagrama de estrutura são o módulo, a chamada, a área de dados compartilhados e os parâmetros. Os dois primeiros dizem respeito às unidades ativas (elementos funcionais ativos) do processo e os dois últimos modelam os elementos funcionais passivos sobre os quais atuam essas unidades.

1.1.1. O Módulo

Um módulo é definido como uma coleção de instruções de programa —código— com quatro atributos básicos:

1. entrada e saída — o que é recebido pelo módulo e o que é por ele produzido
2. funcionalidade — o que é feito com a entrada para que a saída seja produzida
3. lógica — o conjunto de operações e controles que permite ao módulo exercer sua funcionalidade
4. área de dados internos — espaço privado de trabalho do módulo, contendo dados para os quais apenas ele faz referências

Entrada e saída são, respectivamente, as informações que um módulo necessita e fornece.

A funcionalidade de um módulo é o que ele faz para, a partir da entrada, produzir a saída.

Lógica constitui a codificação de procedimento ou algoritmo que executa a funcionalidade.

Dados internos representam a informação contida na área interna de trabalho de um módulo à qual somente o módulo faz referência. Lógica e dados internos representam uma visão interna do módulo.

Um módulo também tem outros atributos: ele possui um nome, pelo qual será referenciado. Ele pode usar ou ser usado por outros módulos.

Módulo é uma unidade de código pertencente a uma estrutura hierárquica que representa o controle do processo. Vale ressaltar que somente um módulo pode estar ativo em um mesmo instante no processo —embora vários processos possam estar ativos concorrentemente. As linguagens de programação possuem diferentes implementações de módulo, cada uma com nomes diferentes: "procedure" em Pascal e Modula-2, "function" em Pascal, "subroutine" em Assembler etc.

1.1.2. Representação Gráfica de Um Módulo

Em um diagrama de estrutura hierárquica, o módulo é representado por um retângulo dentro do qual está contido seu nome, como mostrado na figura 1.1.

O nome do módulo é uma declaração sucinta de sua funcionalidade, isto é, o que ele faz cada vez que é chamado. Não chamamos o módulo de "Rotina de Controle" ou "Rotina de Processamento" porque isto referencia apenas parte de sua lógica e não constitui necessariamente indicação global de sua funcionalidade.

Um módulo pré-definido é mostrado graficamente através da adição de linhas internas e paralelas às linhas verticais —veja figura 1.2. Representa um módulo —ou pequeno subsistema— que não terá de ser reescrito porque já existe em sistemas ou em bibliotecas de aplicações.

Vale ressaltar que muitas instalações utilizam o mesmo símbolo para módulos para representar macros e rotinas próprias do sistema operacional.

1.1.3. Conexões Entre Módulos — Chamadas

Módulos não são herméticos. Um sistema não é uma anarquia de caixas mal humoradas, mas um conjunto de módulos organizados dentro de uma hierarquia, cooperando para realizar algum trabalho.

O símbolo que une os módulos é uma seta mostrando o "alvo" ou, mais precisamente, uma chamada de um módulo.

Considere a figura 1.3. Essa figura diz que:

- A - chama (invoca, executa) B, transferindo-lhe o controle do processamento
- B - executa sua funcionalidade
- B - devolve o controle do processamento a A imediatamente após a instrução na qual A chamou B

Em outras palavras, a seta mostra uma chamada de uma sub-rotina normal, com a direção da seta mostrando qual módulo chama qual.

Uma chamada é, portanto, uma ativação de um módulo. Quando o módulo chamador efetua uma chamada a um subordinado, o chamador torna-se inativo. Conforme já foi mencionado, quando o módulo subordinado completa seu trabalho, o controle é retornado para o chamador no ponto do programa imediatamente após a chamada do subordinado.

Entretanto, deve-se notar que a chamada difere de um comando habilita/inabilita de um esquema de atividades. O módulo chamador torna-se inativo quando a chamada é realizada e permanece nesse estado até que o módulo chamado devolva o controle do processamento, após ter realizado sua funcionalidade. Já uma atividade, responsável por

habilitar ou inabilita uma outra atividade, produz, por exemplo, um comando de habilitação, permanecendo ativa e podendo inabilita a atividade chamada em qualquer instante.

E, ainda, interessante observar outra característica do diagrama de estrutura hierárquica. Na figura 1.3 pode ser verificado que o diagrama não diz praticamente nada a respeito da codificação de A ou de B. A possui uma instrução CHAMAR B, mas essa instrução pode estar dentro de um "IF" ou dentro de condições muito complexas. Pode ser que existam 10, 27 ou até 37 instruções "chamar B" dentro de A. A seta não diz qualquer coisa a respeito da codificação de A —tampouco de B—, nem mesmo diz quantas vezes A chama B ou se em uma certa execução A chamará B ou não. Tudo que verificamos é que A é capaz de chamar B e que B está subordinado a A.

1.1.4. A Comunicação Entre Módulos

Na figura 1.3 A está chamando B sem comunicar-se com B.

Módulos podem comunicar-se através de parâmetros.

Parâmetros representam dados transmitidos entre módulos quando uma chamada é executada. Dados podem ser passados do módulo chamador para o módulo chamado e/ou vice-versa.

Parâmetros são modelados como pequenas setas —alocadas próximas à chamada apropriada— que apontam no sentido em que os dados fluem (vide figura 1.4). No extremo oposto à seta um pequeno círculo é desenhado:

Um círculo aberto - representa dados que serão processados.

Um círculo cheio - representa informação de controle que influenciará a lógica do módulo chamador.

Todos parâmetros devem ser nomeados e, esse nome, deverá ser o nome utilizado pelo módulo chamador. Esse aspecto implica em nomes de dados mais específicos. Módulos da parte mais baixa da hierarquia, freqüentemente, executam ações utilitárias genéricas que, normalmente, "afastam-se" do domínio da aplicação.

1.1.5. A Área de Dados Compartilhados

Essa área contém dados que são utilizados por mais de um módulo. Os dados são acessíveis por todos os módulos que têm acesso à área.

A área de dados compartilhados é modelada por um retângulo desenhado na parte inferior de cada módulo que utiliza os dados (vide figura 1.5). O nome da área é escrito dentro desse retângulo.

1.1.6. A Especificação de Módulos

O diagrama de estrutura hierárquica mostra somente a estrutura hierárquica geral de um processo e, deliberadamente, omite quase todos os detalhes de procedimento. No entanto, o programador que usará o diagrama para realizar sua codificação está muito interessado nesses detalhes de procedimento. Como, então, essas informações podem ser fornecidas ao programador?

Todo módulo existente em um diagrama de estrutura hierárquica deve possuir uma especificação —especificação de módulo. O objetivo é, para cada módulo definido, haver uma especificação procedimental correta e precisa das ações executadas pelo módulo —ainda que, nesse estágio, não

deva ser usada uma linguagem de programação.

Muitos dos módulos serão derivados diretamente de porções do Modelo da Essência. As ferramentas procedimentais [1, cap.8, vol. 1] adequadas para descrever o comportamento detalhado das atividades primitivas do Modelo do Comportamento —tais como linguagem estruturada, pseudocódigo etc.— deverão ser empregadas.

A quantidade de detalhes envolvidos na especificação deverá depender, apenas, da linguagem alvo. Por exemplo, haverá pouca utilidade em usar pseudocódigo para especificar as ações de um módulo que será desenvolvido em uma linguagem de programação de alto nível, como Pascal; entretanto, se o módulo for desenvolvido em uma linguagem de programação de baixo nível, tal como "Assembler", os detalhes poderão ser de fundamental importância.

Sem uma especificação de módulo, não há como dar manutenção ao sistema.

Suponha que você seja chamado para investigar um erro num sistema escrito há muitos anos por uma pessoa. Após árduas investigações, você descobre que o problema deve estar numa certa área da codificação —suponhamos, em um único módulo. Sua linha de pensamento deve continuar como segue:

"Se eu pelo menos soubesse o que este módulo deveria fazer. Não parece haver nenhuma documentação explicativa em nenhum lugar. Suponho que terei de deduzir a partir da codificação!"

Essa armadilha, que continua preocupando o profissional de manutenção de sistemas, seria evitada se o primeiro programador tivesse feito uma especificação de módulo. Na pior das hipóteses, se a codificação que está sofrendo manutenção for um desastre total, o módulo poderá ser totalmente reescrito a partir de sua especificação.

1.1.7. A Especificação de Dados

Não apenas os módulos, mas também os parâmetros e as áreas de dados compartilhados devem ser especificados.

Analogamente à especificação de módulos, muitas especificações de dados podem ser aproveitadas diretamente das várias porções do Modelo da Essência alocadas ao Modelo da Implementação. Haverá, entretanto, alguns detalhes de implementação, não tocados durante a modelagem da essência, que deverão ser levados em consideração.

Inicialmente, a representação de cada elemento de dado deverá ser definida. O Modelo da Essência específica, apenas, domínio, unidade, precisão e, às vezes, formato de um elemento de dado. Deve-se, agora, escolher uma representação interna apropriada para o elemento.

Linguagens de programação de alto nível, normalmente, não requerem esse tipo de detalhe para sua utilização. Entretanto, se tratamos com uma linguagem de baixo nível, a representação interna deve ser explicitada na especificação dos elementos de dados.

Outro aspecto refere-se à ordenação de elementos de dados em alguma representação estruturada. Analogamente, as linguagens de programação de alto nível permitem que os componentes de uma estrutura sejam referenciados pelo nome e, portanto, tornam a ordenação irrelevante. Já nas linguagens de baixo nível, pode haver a necessidade de definir-se o ponteiro base da estrutura de dados. A quantidade de espaço necessário a cada item é especificada pela representação escolhida. Se a ordem dos elementos é importante, a estrutura de dados deve possuir essa informação.

Por fim, áreas de dados compartilhados podem necessitar da especificação de valores iniciais, que serão fornecidos quando o sistema é "carregado" e executado ou está em fase de inicialização.

Para exemplificar, podemos pensar em uma linguagem de programação de baixo nível, como "Assembler", que possui um módulo que é sempre chamado quando o processo é ativado.

1.2. ANÁLISE DE TRANSAÇÃO

A alocação de várias atividades primitivas a um mesmo processo é freqüentemente necessária ou conveniente devido a relacionamentos de tempo que possam existir entre elementos do Modelo da Essência.

Por exemplo, se as taxas desejadas para acesso à Unidade Central de Processamento (UCP) por determinadas atividades primitivas, já alocadas a um processador, são bastante semelhantes e essas atividades são compatíveis com uma utilização seqüencial da UCP.

Outro exemplo pode referir-se a restrições associadas ao número máximo de processos concorrentes —pensando em atividades que possuam um processamento pequeno, em que a execução do pouco código associado ao conjunto dessas atividades será mais econômica se essas atividades estiverem todas alocadas ao mesmo processo.

Nesses casos, é suficiente criar um módulo para cada atividade primitiva do Modelo da Essência e subordiná-los a algum outro módulo "gerente".

Vale ressaltar que, nessa alocação de atividades primitivas a um mesmo processo—trabalho realizado durante o Modelo da Configuração de Processos—, são acrescentadas atividades não-essenciais. Essas, do mesmo modo que as atividades primitivas do Modelo da Essência, serão transformadas em módulos e estarão, também, subordinadas a algum módulo "gerente".

Um caso interessante de alocação ocorre quando várias atividades primitivas do Modelo da Essência possuem um relacionamento entre os dados de entrada tal que, de acordo com a informação inicial, determina-se qual a atividade que deve realizar o processamento desses dados.

Considere a situação ilustrada pela figura 1.6, onde uma atividade não-essencial recebe um fluxo agregado que pode associar-se a várias transações distintas, determina o tipo de transação —pela inspeção dos dados de entrada— e endereça a entrada para o módulo de transação apropriado. Essa atividade chama-se um "Centro de Transação" e o procedimento que permite, analisando o diagrama em rede (DFD), identificar um "Centro de Transação" e as atividades primitivas que processam as diferentes transações recebidas pelo Centro de Transação chama-se "Análise de Transação".

Em outras palavras, a partir dessa análise é possível construir um diagrama de estrutura hierárquica baseada num "Centro de Transação". Trata-se de procedimento extremamente usado para otimizar a entrada de dados de um sistema —um canal de entrada para vários fluxos do Modelo da Essência.

Esse procedimento consiste em criar-se um módulo para obter a transação e, para cada tipo de transação, criar-se um módulo para seu processamento. Todos esses módulos estarão subordinados a um outro módulo "gerente" hierarquicamente mais alto, como na figura 1.7.

Podemos imaginar um departamento que possua uma secretaria, um almoxarifado e um setor de processamento de dados. Se pudéssemos melhor aproveitar a distribuição de funções de acordo com a chegada de informações no departamento, aplicaríamos uma estratégia de Análise de Transação: determinaríamos um novo setor —que poderia ser uma parte da secretaria, por exemplo, uma das secretárias— responsável pelo recebimento das informações e distribuí-las segundo o tipo de informação para os setores adequados, a fim de que esses executem o processamento que lhes cabe.

A Análise de Transação tem duas funções principais:

19. Dividir um esquema de atividades primitivas (diagrama de rede), de grande porte e alto grau de complexidade intrínseca, em esquemas menores, um para cada transação. Esses esquemas menores permitirão detalhamento complementar, pela Análise de Transformação —que veremos mais à frente nesta seção— também em termos de diagramas de estrutura hierárquica.

20. Pode ser usada para combinar diagramas de estrutura hierárquica individuais de transações separadas em um diagrama de estrutura maior e mais flexível em relação a alterações propostas pelos usuários.

Uma transação, de forma geral, inicia-se por um estímulo incidente sobre um sistema que possui um conjunto de atividades a ser realizadas internamente para responder ao estímulo. Exemplos de estímulos são um sinal para um veículo espacial disparar seus retrofoguetes, um alarme de temperatura em um reator ou uma interrupção na medição de tempo em um sistema operacional. As atividades associadas a essas transações poderiam ser, respectivamente, o disparar dos retrofoguetes do veículo espacial, o desligar do reator e o início de uma nova medida de tempo.

A definição de transação escrita acima é suficientemente ampla para ser aplicada a sistemas de tempo real, tais como controles de processos ou sistemas operacionais. Entretanto, uma definição mais abstrata que é válida para a maioria dos outros propósitos é: uma transação é uma coleção de dados de entrada inter-relacionados processada por um conjunto específico de atividades em um sistema, culminando com a geração de dados de saída também inter-relacionados e coerentes com os dados de entrada.

O método de Análise de Transação é claramente particionado, de modo que qualquer mudança no processamento de um tipo de transação não afeta o processamento de qualquer outro tipo de transação.

A penalidade que aparentemente teremos de pagar por essa codificação clara é termos códigos idênticos em módulos distintos, já que muitos tipos de transação requerem processamento semelhante. Esse problema pode ser eliminado pela aplicação do "fatoramento" ("fan-in"), técnica de otimização a ser abordada na próxima seção.

1.2.1. Centro de Transação em Sistemas "On-Line"

Um tipo de centro de transação freqüentemente encontrado em sistemas "on-line" é aquele nos quais os usuários têm um menu de opções para seleção.

Um exemplo seria um sistema para direcionar um barco condutor de mísseis a partir de um terminal remoto. Veja tabela 1.1 e a figura 1.8 contendo o centro de transação para implementação desse sistema.

Uma característica interessante do centro de transações da figura 1.8 é que o módulo do topo chama cada um dos módulos de transação sem utilizar nenhum dado. Isto não é um erro! Cada subsistema de processamento de transação é responsável pela obtenção de sua própria entrada e fornecimento de sua própria saída. Desse modo, separando as respectivas entradas e saídas de cada subsistema, obtém-se flexibilidade muito maior do que a obtida se o módulo do topo coletasse a entrada e a encaminhasse para o módulo de transação apropriado, recebendo a saída do

módulo de transação. Esse método requeria que o módulo do topo se preocupasse com a sintaxe de quase todas as informações no sistema, mas como o módulo no topo não necessita dessas informações, teríamos um maior acoplamento entre os diferentes módulos da hierarquia, característica a ser evitada. Além disso, seria difícil modificar o sistema para permitir que cada subsistema tivesse sua própria comunicação com o usuário ou obtivesse dados de sua própria base de dados.

1.2.2. Identificação de Transações em Esquemas de Atividades

Muitas vezes é simples reconhecer transações, centros de transação e atividades de processamento de transação em um esquema de atividades através do formato desse esquema. Sempre que um fluxo agregado incide em uma atividade que determina suas características e o envia para uma atividade relacionada com as características identificadas, pode-se ter certeza que foi localizada uma transação entrando e um centro de transação. Entretanto, deve-se ressaltar que a construção do Centro de Transação, normalmente, não é fácil.

O esquema de atividades para um centro de transação de campos, bem como seu diagrama de estrutura são representados nas figuras 1.9(a) e 1.9(b). Foi assumido que todos os campos validados e formatados seriam coletados dentro de um registro único. A atividade "Distribuir Campos por cada Controle" é o Centro de Transação, o qual envia cada campo para a atividade apropriada, que o valida e formata.

Vale ressaltar que, utilizando a heurística proposta por Ward & Mellor[1, vols. I e II] de separar explicitamente controle de operação, focalizando paralelamente atividades de controle e atividades operacionais, obtém-se facilmente o centro de transação. Basta verificar que cada centro de transação é responsável pela obtenção dos dados de entrada necessários a seu processamento — não havendo necessidade de uma atividade distinta para obter os dados de entrada — e, naturalmente, é responsável pela elaboração de sua própria saída.

Um exemplo para ilustrar essa heurística é apresentado na figura 1.10. O centro de transação desse esquema está, obviamente, na atividade de controle "Invocar Subsistema Apropriado". Para transformar esse esquema no diagrama de estrutura da figura 1.8 é necessário apenas considerar a atividade de controle como um centro de transação.

1.2.3. Observação Final

Deve, ainda, ser ressaltado que Análise de Transação e Análise de Transformação podem ser combinadas utilizando-se a primeira para criar uma estrutura de alto nível hierárquico e, a partir desta, usando-se a segunda para criar uma sub-hierarquia que processará cada tipo de transação. A figura 1.11 mostra uma das sub-hierarquias adicionadas ao diagrama de estrutura da figura 1.7 e, a seguir, a heurística baseada na "Análise de Transformação" será apresentada.

1.3. ANÁLISE DE TRANSFORMAÇÃO

O esquema de atividades apresentado na figura 1.12 faz parte do Modelo da Configuração de Processos associado a uma porção do Modelo da Essência de um sistema computacional. (Convém ressaltar que esse esquema de atividades pode representar o processamento de uma transação determinada pela Análise de Transação). Esse tipo de esquema pode ser transformado em um diagrama de estrutura utilizando-se um procedimento conhecido como "Análise de Transformação".

O procedimento de Análise de Transformação requer a identificação do "Centro de Transformação" do esquema, que corresponde à atividade que possui a entrada de dados em sua forma mais essencial e produz a saída, também, na forma mais essencial —isto é, desprovida de elementos de implementação.

Procedemos, então, excluindo atividades que validam a entrada e atividades que mudam dados na forma de implementação para dados na forma essencial —ou vice-versa. Resta-nos, por fim, uma ou poucas atividades que são boas candidatas a se tornarem o Centro de Transformação ou que lhe dão origem. A atividade "Calcula Valor Médio do pH", na figura 1.12, parece-nos uma escolha razoável.

As atividades periféricas ao Centro de Transformação formam um ou mais ramos que são classificados como:

19. "AFERENTES": conduzem dados a ser processados pelo Centro de Transformação

20. "EFERENTES": conduzem dados produzidos pelo Centro de Transformação.

A figura 1.13 mostra o esquema da figura 1.12, indicando os ramos aferentes e eferente.

E a principal estratégia para projetar sistemas "balanceados".

Um sistema "balanceado" é aquele cujos módulos superiores lidam mais com dados de natureza lógica do que com dados de natureza física.

Em seu lado aferente, o dado é editado e desbloqueado, deixando de depender da forma com que ele entrou no sistema. Em seu lado eferente, próximo ao centro de transformação, o dado é bastante independente de qualquer formato especial de relatório que possa ser solicitado pelo usuário. Enfim, o Centro de Transformação é isolado dos ambientes de entrada e saída através da alocação desses em uma sub-hierarquia separada; e os módulos de mais alto nível são isolados dos detalhes de obter e enviar dados para o ambiente externo dos módulos de mais baixo nível, uma vez que aqueles apenas vêem os resultados das atividades dos outros.

Usando-se esse tipo de heurística, obtém-se sistemas de manutenção mais fácil já que possuem um acoplamento entre módulos bem definido e uma grande adaptabilidade a mudanças específicas —especialmente mudanças nos dispositivos físicos ou formatados.

A Análise de Transformação é composta por quatro passos:

1. encontrar as funções primitivas essenciais do esquema de atividades;
2. converter o esquema de atividades em um diagrama de estrutura hierárquica preliminar;
3. refinar esse diagrama de estrutura, utilizando os critérios apresentados na próxima seção;
4. verificar se o diagrama de estrutura produzido está de acordo com os requisitos modelados no esquema de atividades original.

1.3.1. Passo 1: Identificar o Centro de Transformação

O Centro de Transformação é a parte do esquema de atividades que contém as funções primitivas essenciais do sistema e é independente da implementação particular das entradas e saídas.

Pode ser encontrado de duas maneiras.

A primeira maneira para identificar o centro de um esquema de atividades é através da observação. Por exemplo, na figura 1.13, não sabemos seguramente, mas é razoável supor que a atividade F5 seja ou faça parte do centro de transformação.

Uma segunda maneira, melhor, para encontrar o Centro de Transformação é identificar o centro do esquema de atividades, pela eliminação de seus ramos aferentes e eferentes. Pode-se fazer isto através de três passos:

1º. Trace cada curso aferente partindo do lado externo do esquema de atividades em direção ao centro. Marque o fluxo de dados que representa a entrada em sua forma mais lógica. Em outras palavras, marque a etapa na qual a entrada foi completamente refinada, mas que ainda não foi usada para processamento.

2º. Trace cada curso eferente do lado externo do esquema de atividades em direção ao centro. Marque o fluxo de dados que represente a saída em sua forma mais lógica. Em outras palavras, marque a etapa na qual a saída foi produzida, mas ainda não está formatada.

3º. Verifique que a(s) atividade(s) restante(s), isto é, aquela(s) atividade(s) que não faz(em) parte dos ramos aferente e eferente, forma(m) um conjunto central que contém, de alguma forma, o Centro de Transformação.

Usa-se, freqüentemente, o método de retorno ao "mundo ideal", desenvolvido no Modelo da Essência, para determinar quais atividades primitivas são boas candidatas a ser o Centro de Transformação —ou dar-lhe origem— no esquema de atividades. Para utilizarmos esse método devemos tentar identificar o esquema de atividades abordado no Modelo do Comportamento, evitando as atividades não-essenciais incluídas durante a modelagem da implementação.

Por exemplo, se os dados de entrada, uma vez validados, nunca dessem origem a erros, não existiria a necessidade de processos internos (atividades não-essenciais) de correção de erros. Se o usuário não se importasse com o formato do relatório, não haveria necessidade de outros processos de formatação. Logo, restariam apenas atividades que poderiam qualificar-se para Centro de Transformação.

1.3.2. Passo 2: Produzir Um Diagrama de Estrutura Preliminar

A principal diferença entre um esquema de atividades em rede e um diagrama de estrutura hierárquica é que o diagrama representa uma relação de subordinação. Em um esquema de atividades não existem superiores e subordinados. A hierarquia dos níveis de um conjunto associado a um esquema de atividades representa uma hierarquia baseada num princípio de equivalência e não num princípio de subordinação. No diagrama de estrutura hierárquica, a hierarquia de módulos representa tanto os controles como alguns detalhes de processamento.

As atividades são como trabalhadores numa comunidade, onde cada um realiza seu trabalho e envia o produto para o próximo trabalhador na linha. A aplicação da Análise de Transformação introduz o conceito de hierarquia. A comunidade terá chefes e, em particular, um comandante para toda a organização.

No mundo real, nós teríamos duas escolhas acerca de onde encontrar esse comandante. Poderíamos promover alguém de dentro da organização ou poderíamos contratar um novo chefe de fora da organização.

A Análise de Transformação nos dá essas mesmas duas escolhas.

Uma atividade do Centro de Transformação pode destacar-se como um comandante potencial. Um indício para a escolha de uma atividade do conjunto é selecionar aquela que realiza pouco processamento mas executa um grande número de ações, coordenando o trabalho das outras atividades. Outro indício é a localização geométrica da atividade. Uma atividade central, com vários fluxos de dados incidentes e emergentes pode, provavelmente, qualificar-se para comandante.

Pode haver duas ou três atividades concorrendo para a promoção. Nesse caso, deve-se utilizar uma de cada vez, para saber qual delas proporciona o melhor projeto ("design") inicial.

Entretanto, é possível que não haja nenhum candidato potencial adequado para assumir o posto de comandante. Então, deve ser escolhido um comandante que não esteja presente no esquema de atividades.

Nessa modelagem, criamos um módulo de controle hierarquicamente mais alto e um segundo nível contendo o módulo para o Centro de Transformação e os módulos referentes aos ramos aferente e eferente do esquema de atividades —esse caso está ilustrado na figura 1.14.

Após a hierarquia estabelecida, criamos, caso haja necessidade, as sub-hierarquias para o segundo nível do diagrama. Por exemplo, se a transformação central é formada por apenas uma atividade, então, basta apenas aquele módulo para representá-la. No entanto, se existirem várias atividades nessa transformação central, elas deverão vir a formar uma nova hierarquia —subordinada ao Centro de Transformação—, ou seja, o terceiro nível no diagrama. A figura 1.15 exemplifica a transformação de duas atividades, cada uma em módulos separados.

Recapitulando o passo dois:

SE há um bom candidato para comandante

ENTÃO escolha o comandante e subordine a ele todas as demais atividades

SENAO encontre um novo comandante e subordine a ele o conjunto central de atividades e os ramos aferente e eferente

Os passos 3 e 4 serão apresentados após termos conhecimento de características de refinamento de diagramas de estrutura —como "acoplamento" e "coesão"— abordadas na próxima seção. Deve-se entender **refinamento** como a prática de se melhorar o diagrama de estrutura hierárquica visando obter uma organização otimizada do código.

2. REFINAMENTO DE UM DIAGRAMA DE ESTRUTURA

2.1. MOTIVAÇÃO:

Um dos princípios fundamentais da modelagem hierárquica é que um grande sistema deveria ser particionado em módulos "mais" simples. No entanto, é vital que essa partição seja feita de tal maneira que os módulos sejam tão independentes quanto possível —esse é o critério de baixo acoplamento— e que cada módulo possua uma única funcionalidade —esse é o critério de alta coesão.

Naturalmente, precisamos mais do que critérios de baixo acoplamento e alta coesão para projetar sistemas de fácil manutenção e isso porque esses dois critérios, se mal empregados, podem entrar em conflito. Por exemplo, se nos preocupássemos somente com o acoplamento, produziríamos sempre sistemas contendo um único módulo. Nesse caso, não poderíamos dizer que esse módulo esteja fortemente acoplado a outros, simplesmente porque não haveria outros módulos! Entretanto, a coesão daquele único módulo provavelmente seria muito baixa, o que violaria o critério de alta coesão. Assim sendo, precisamos levar em consideração outras diretrizes dentro de nossa abordagem visando melhorar diagramas de estrutura hierárquica.


2.2. ACOPLAMENTO:

Acoplamento é o grau de interdependência entre dois módulos.

Módulos são acoplados devido a suas interconexões. A conexão entre módulos, como já vimos, pode ter a forma de um relacionamento de chamada, de parâmetros passados entre módulos ou de área de dados compartilhados.

Também é possível acoplar módulos através da representação de um módulo que transfere o controle para outro módulo por meio de um comando "goto". Entretanto, esse tipo de acoplamento viola a prática de um bom projeto ("design") e não o levaremos em consideração.

Estabelecemos que o acoplamento fraco entre módulos significa que o sistema foi bem projetado. Examinemos agora cinco tipos diferentes de acoplamento que podem ocorrer entre dois módulos. Em ordem crescente de acoplamento, eles são[3]:

- | | |
|----------------------------|-----------------|
| 1. Acoplamento de Dados | boa, ou solta |
| 2. Acoplamento de Imagem | |
| 3. Acoplamento de Controle | |
| 4. Acoplamento Comum | |
| 5. Acoplamento de Conteúdo | ruim, ou rígida |
- 

1.4. BIBLIOGRAFIA

1. "Structured Development for Real-Time Systems"
Volume 1: "Introduction & Tools"
Volume 2: "Essential Modeling Techniques"
Volume 3: "Implementation Modeling Techniques"
Paul T. Ward & Stephen J. Mellor
(Yourdon Press, 1985)
2. Projeto Estruturado de Sistemas
Meilir Page-Jones
(McGraw-Hill, 1988)

2.2.1. Tipos de Acoplamento

2.2.1.1. Acoplamento de Dados

Dois módulos são acoplados por dados se eles se comunicam por parâmetros, sendo cada parâmetro um único elemento de dado ou uma tabela homogênea de elementos de dado, isto é, uma tabela na qual cada entrada contém o mesmo tipo de informação. Um bom exemplo para tal forma de acoplamento é apresentado na figura 2.1.

Acoplamento de dados é a comunicação de dados necessária entre módulos. Uma vez que módulos têm que se comunicar, a ligação de dados é inevitável e é inofensiva desde que mantida a taxas mínimas.

No exemplo da figura 2.1, não podemos ficar sem nenhum dos quatro grupos de dados: "Quantidade Emprestada", "Taxa de Juros", "Prazo" e "Taxa de Reembolso". Por outro lado, não precisamos de nenhum elemento adicional, como "Nome do Cliente", que adicionaria uma complicação extra sem ter utilidade para o módulo "Calcular Reembolso da Hipoteca".

As interfaces entre módulos devem ser mantidas apenas com os relacionamentos necessários.

2.2.1.2. Acoplamento de Imagem

Dois módulos são ligados por imagem se eles se referem à mesma estrutura de dados. Estrutura de dados é um conjunto de elementos de dados, tal como um registro constituído por vários campos. A figura 2.2 apresenta três módulos ligados por imagem.

No exemplo da figura 2.2, o parâmetro "Registro de Aluguel de Carro" compreende vários campos: "Número de Licença", "Sócio do Clube Merlin", "Número do Clube Merlin", "Gasolina Usada", "Tipo de Carro", "Quilômetros Rodados", "Dias de Uso", entre outros.

Apesar do módulo "Calcular Taxa de Aluguel Básico" requerer somente os últimos três campos mencionados, ele recebe toda a estrutura de dados "Registro de Aluguel de Carro". Qualquer alteração nesse registro, tanto em formato como em estrutura, afetará todos os módulos que a ele se refiram, mesmo os módulos que não utilizem os campos modificados.

O acoplamento por imagem apresenta dois problemas essenciais:

1. Ele cria dependência entre módulos anteriormente não-relacionados.
2. Tende a expor o módulo a mais dados do que ele necessita, com possíveis conseqüências desastrosas.

O primeiro problema é inevitável se há a necessidade de se utilizar uma estrutura de dados. Abordaremos melhor o segundo problema adiante.

2.2.1.3. Acoplamento de Controle

Dois módulos são acoplados por controle se um passa para o outro um grupo de dados destinados a controlar a lógica interna do outro. A figura 2.3 mostra dois módulos ligados por controle.

O valor do sinal "O Que Fazer" indica ao módulo "Controle Entrada/Saída do Sistema" qual(is) registro(s) ler, como segue: valor 1 indica "Obter Próximo Registro Mestre"; o valor 2, "Obter Próximo

Registro de Transação"; o valor 3, "Obter Ambos"; o valor 4, "Imprima Cabeçalhos de Páginas etc..

Na figura 2.3, o módulo "Juntar Registros de Clientes" decide, explicitamente, qual parte da lógica do módulo "Controle de Entrada/Saída do Sistema" usar. Para que o módulo chamador possa tomar tal decisão, ele precisa saber como a lógica do módulo subordinado chamado está organizada e qual sua natureza.

Em módulos acoplados por controle, o módulo subordinado não é uma caixa preta. Na verdade, um controle passado de um módulo comandante para um módulo subordinado geralmente implica que o módulo receptor é desastrosamente uma caixa branca.

Se um indicador de controle está se dirigindo de um módulo subordinado para o módulo comandante —ilustrado na figura 2.4—, então, outro tipo de partição errônea aparece: **invasão de autoridade**, que significa que um módulo subordinado dá ordens a um comandante.

No exemplo da figura 2.4, o módulo "Encontrar Nome do Cliente" diz ao seu módulo comandante "Produzir Relatório de Pagamento de Cliente" para imprimir uma mensagem de erro sempre que um número não for encontrado. Isso desordena a hierarquia; é equivalente a um trabalhador dizer: "Patrão, faça isto" ou "Patrão, não faça isto". Outro problema é que o módulo "Encontrar Nome do Cliente" assume que qualquer módulo que o chame é capaz de emitir mensagens de erro.

O acoplamento de controle não é sempre tão explícito como mostrado acima.

Algumas vezes está disfarçado de uma forma conhecida como **acoplamento híbrido**[4] —Yourdon e Constantine usaram o termo "acoplamento híbrido" para o comando "ALTER" do COBOL. Define-se o termo aqui em concordância com o seu uso corrente. Esse acoplamento resulta na indicação de vários significados para várias partes do domínio de um grupo de dados, o que tende a causar problemas intensos de manutenção. Se um grupo de dados representasse um animal, então, o acoplamento híbrido resultaria em uma criatura tendo a frente de um cisne, as costas de um camelo e os rins de um rinoceronte. Um exemplo ilustrativo desse problema é apresentado em [3, pág. 119].

2.2.1.4. Acoplamento Comum

Dois módulos possuem acoplamento comum se eles se referem a mesma área de dados —isto é, possuem uma área de dados compartilhados (figura 2.5). Os termos "área de dados compartilhados", "área comum" e "área global" possuem o mesmo significado.

O acoplamento comum não é aconselhável por cinco razões —apesar de ser difícil que dois módulos com acoplamento comum sofram de todas as cinco razões:

19. Um erro, ou má programação, em qualquer módulo que use uma área global pode aparecer em qualquer outro módulo que use aquela área, porque dados globais não residem na área local (protegida) de um módulo. Uma área comum é uma selva na qual um conjunto de dados pode ser processado e deformado a qualquer momento.

20. Módulos que referenciam dados globais geralmente o fazem através de nomes explícitos. Por exemplo, um módulo editor do campo "Número de Telefone" pode receber seu número de telefone de um conjunto de dados chamado "Tel-N". Esse módulo agora está preso ao nome "Tel-N". Ele não pode editar um número de telefone com outro nome além de "Tel-N" —tanto em outros lugares desse sistema como em qualquer outra aplicação. Apenas se tivéssemos sorte o suficiente para encontrar uma outra aplicação usando mesmo nome "Tel-N", poderíamos (re-)usar esse módulo.
30. Áreas globais ou comuns podem, algumas vezes, sofrer abusos drásticos como, por exemplo, quando módulos diferentes usam a mesma área para armazenar diferentes grupos de informações. Há situações em que uma área de dados compartilhados, por exemplo, com nome "Status Produção", acessada por dois módulos diferentes, possui significados diferentes de acordo com os módulos que a acessam. Para um módulo, "Status Produção" representa "estoque insuficiente" e para outro módulo "Status Produção" representa "não existe tal número de material". Nesses casos, os esforços de manutenção —especialmente para tirar erros— são tão cabeludos como um aperto de mão de um lobisomem.
40. Programas com muitos dados globais são extremamente difíceis de entender pelos profissionais de manutenção, porque é difícil descobrir quais dados são usados por um certo módulo. E especialmente difícil falar qual a ligação real entre esses módulos, já que é necessário descobrir se existem dados ligando os módulos e quais deles são compartilhados.
50. Da mesma forma que é difícil descobrir quais dados precisam ser mudados quando um módulo tem de ser alterado, também é difícil descobrir quais módulos devem ser alterados quando um dado é modificado. Por exemplo, se um registro numa área global tem de ser mudado de 96 para 112 bytes, muitos módulos poderão ser afetados. Mas quais? Sem que haja uma referência cruzada, torna-se necessário verificar todos os módulos do sistema. Isso demanda tempo e, portanto, dinheiro.

A modularidade deve restringir um elemento de dado particular a um só módulo tanto quanto possível ou, no máximo, a um pequeno grupo de módulos. O excesso de uso de dados comuns degrada essa idéia de modularidade, pois deixa dados abandonarem os limites estritos de um módulo.

2.2.1.5. Acoplamento de Conteúdo

Dois módulos apresentam acoplamento de conteúdo —ou patológico— se um faz referência ao interior do outro: por exemplo, se um módulo desvia a seqüência de instruções para dentro de outro módulo ou se um módulo altera um comando do outro.

Tal acoplamento torna o conceito de módulos-caixa-preta sem sentido, já que força um módulo a conhecer explicitamente a natureza e a organização do conteúdo de outro módulo, isto é, sua implementação. Excluindo o uso do comando "goto", apenas linguagens montadoras permitem o uso de tais práticas; a maioria das linguagens de alto nível não oferecem meios de implementar o acoplamento de conteúdo.

O acoplamento de conteúdo nos traz de volta ao mundo desorganizado da programação não modular.

2.2.2. Obtendo Um Baixo Acoplamento

Usar acoplamento como técnica de refinamento significa examinar o acoplamento existente entre dois módulos e modificar o diagrama de estrutura visando minimizá-lo onde for possível, isto é, tornar os módulos tão independentes quanto possível.

Um acoplamento baixo entre módulos indica um sistema bem particionado e pode ser obtido de três maneiras:

1. Agrupando elementos de dado em estruturas de dados;
2. Colocar na área compartilhada parâmetros que atravessam módulos que não os usam;
3. Eliminar acoplamentos de controle.

2.2.2.1. Criação de Estruturas de Dados

Criar estruturas de dados adequadas a partir de elementos de dado determina uma simplificação visual de um diagrama de estrutura, uma vez que basta ao projetista seguir alguns parâmetros para entender o movimento —a navegação— de dados entre os módulos.

Estruturas de dados podem também simplificar o próprio código gerado caso a linguagem de programação que será utilizada possua facilidades para a manipulação de estruturas de dados como unidades.

Para utilização apropriada desse modo de reduzir o acoplamento entre módulos, constrói-se uma estrutura de dados que reúna elementos de dado que sejam globalmente necessários a um grupo de módulos —ou, ao menos, que esses módulos usem a maioria desses elementos de dado.

Entretanto, se há uma estrutura de dados disponível para alguns módulos que utilizam poucos elementos de dados dessa estrutura, os módulos tornam-se bastante acoplados devido à quantidade desnecessária de elementos de dados exposta para cada módulo. Verifique a ilustração desses dois casos nas figuras 2.6(a) e 2.6(b).

Essa idéia de agrupar dados razoavelmente não-relacionados numa estrutura de dados artificial é chamada "empacotamento". Fazendo isso não se obtém nada além de obscuridade desnecessária. Um sinal de que a estrutura de dados não está apropriada ao problema é um nome vago e sem sentido para referenciá-lo. As pessoas empacotam dados porque acham que, de alguma maneira, estão reduzindo o acoplamento. No entanto, empacotamento tem o mesmo efeito terapêutico de um curativo numa espinha.

Vale observar que a aparente simplicidade de "navegação", entre módulos, de estruturas de dados complexas, como se fossem parâmetros simples, apenas mascara a exposição desnecessária a esses módulos, de elementos de dados que não serão por eles utilizados e que podem, sem querer, ser modificados.

Uma outra consideração na criação de estruturas de dados diz respeito à natureza do agrupamento dos elementos de dados.

Os elementos de dados que compõem uma estrutura de dados devem, sempre que possível, possuir algum relacionamento em termos de uma das necessidades que o sistema computacional deverá atender. Um agrupamento arbitrário de elementos de dados —como por exemplo, Dados_Entrada— não contribui muito para a compreensão do diagrama de estrutura hierárquica e pode mascarar a agregação, em um mesmo módulo, de funções que não possuem relação alguma.

2.2.2.2. Utilização de Areas de Dados Compartilhados

Se um elemento de dado ou um conjunto de elementos de dado está disponível para dois ou mais módulos, nenhum deles possuindo uma ligação direta de subordinação —isto é, estão separados—, então, a passagem de parâmetros entre esses módulos tem como consequência direta a navegação de dados por módulos que não os utilizarão —a figura 2.7(a) ilustra essa situação. A esse conjunto de elementos de dados que vagueia pelo sistema, indesejável e sem sentido para a maioria dos módulos pelos quais passa, pode chamar-se de "migrante".

A criação de áreas de dados compartilhados entre um pequeno número de módulos —apresentado na figura 2.7(b)— reduz significativamente o acoplamento no diagrama de estrutura hierárquica.

Entretanto, deve ser ressaltado que o uso de tal solução para reduzir o acoplamento entre módulos é restritivo. Areas de dados compartilhados globalmente, isto é, que tornam disponíveis todos os dados para todos os módulos, determinam um grande acoplamento no diagrama de estrutura hierárquica. Do mesmo modo, tornar disponível, em uma área de dados compartilhados, uma estrutura de dados para um grupo de módulos que utilizam apenas uns poucos dados dessa estrutura constitui um mal uso da área de dados compartilhados.

Uma área de dados compartilhados por um grupo de módulos é um recurso que, além de diminuir o acoplamento entre módulos, protege os módulos que não participam da utilização dessa área de dados compartilhados de detalhes em relação a armazenamento de informação e sobre o mecanismo de manipulação.

2.2.2.3. Eliminação de Acoplamentos de Controle

Alguns parâmetros de controle, tais como aqueles sinais que indicam fim de dados ou falha na localização de dados armazenados, são integrantes inevitáveis do diagrama de estrutura hierárquica e, portanto, não podem ser eliminados.

Contudo, os parâmetros de controle podem, normalmente, ser eliminados de dois modos:

1. Alocando, ao mesmo módulo que determina a condição que conduz a uma determinada ação, a execução correspondente a essa ação; ou
2. Dividindo um módulo chamado, que recebe um parâmetro de controle indicando qual a atividade que deve executar, em dois ou mais módulos que executem atividades atômicas.

O último modo descrito de se eliminar os parâmetros de controle é ilustrado na figura 2.8. O módulo "Apresenta ou Transmite Estados Lógicos" —na figura 2.8(a)— requer um parâmetro de controle, quando ocorre uma chamada, para que internamente ele possa decidir qual função executar. Já os dois módulos independentes "Apresenta Estados Lógicos"

e "Transmite Estados Lógicos", ilustrados na figura 2.8(b), não necessitam do parâmetro de controle para saber qual ação executar. A decisão lógica para determinar a ação a ser executada está escondida no módulo chamador. Além disso, a interface entre os módulos tornou-se mais simples.

2.2.3. Determinação do Tipo de Acoplamento

Dois módulos podem estar acoplados por mais de uma maneira. Neste caso, o acoplamento entre eles é definido pelo pior tipo que apresentam. Por exemplo, se dois módulos são ligados por acoplamento de imagem e comum, caracterizamo-los como de acoplamento comum. Eles ainda serão ligados por imagem, mas isto é sobrepujado por sua ligação de conteúdo.

Outra maneira pela qual se pode avaliar o acoplamento de um projeto ("design") é supor que cada módulo será codificado por um programador diferente:

1. Quanto independentemente podem os programadores trabalhar?
2. Existe algum fato, suposição, convenção ou decisão de implementação sobre os quais mais de um módulo deva estar consciente?
3. Quais as chances e as conseqüências advindas de um fato, uma asserção, uma convenção ou uma implementação mudar?
4. Existe alguma maneira dessa mudança ser isolada a um módulo?

Ao responder a essas perguntas ficará determinado quais prováveis alterações do usuário exigirão manutenção em um grande número de módulos.

2.2.4. Observações Finais

Myers[3] foi o primeiro a desenvolver a escala de acoplamento descrita na seção 2.2.1. Atualmente, a teoria de acoplamento está sendo refinada para diferenciar tipos de acoplamento que encorajem o efeito em cadeia e tipos que tornem as alterações solicitadas pelos usuários difíceis de ser implementadas.

Para resumir, queremos um acoplamento fraco porque:

1. Quanto menos conexões existirem entre módulos, menor a chance do efeito em cadeia — um erro em um módulo aparece como conseqüência de um erro cometido em outro.
2. Queremos ser capazes de trocar um módulo com um mínimo de risco de ter de trocar outro módulo.
3. Queremos que cada mudança solicitada pelo usuário afete o menor número possível de módulos existentes.

4. Enquanto estivermos fazendo a manutenção de um módulo, não queremos nos preocupar com detalhes internos —codificação— de nenhum outro módulo.
5. Queremos que o sistema seja tão simples de ser entendido quanto possível.

No entanto, vale ressaltar que reduzir o acoplamento em detrimento de outro critério de refinamento do diagrama de estrutura hierárquica pode produzir um resultado pior no projeto ("design").

2.3. COESÃO:

Outro meio de avaliar a modularidade de um sistema é olhar para como as ações de um módulo estão relacionadas umas às outras: este é o critério da "COESÃO".

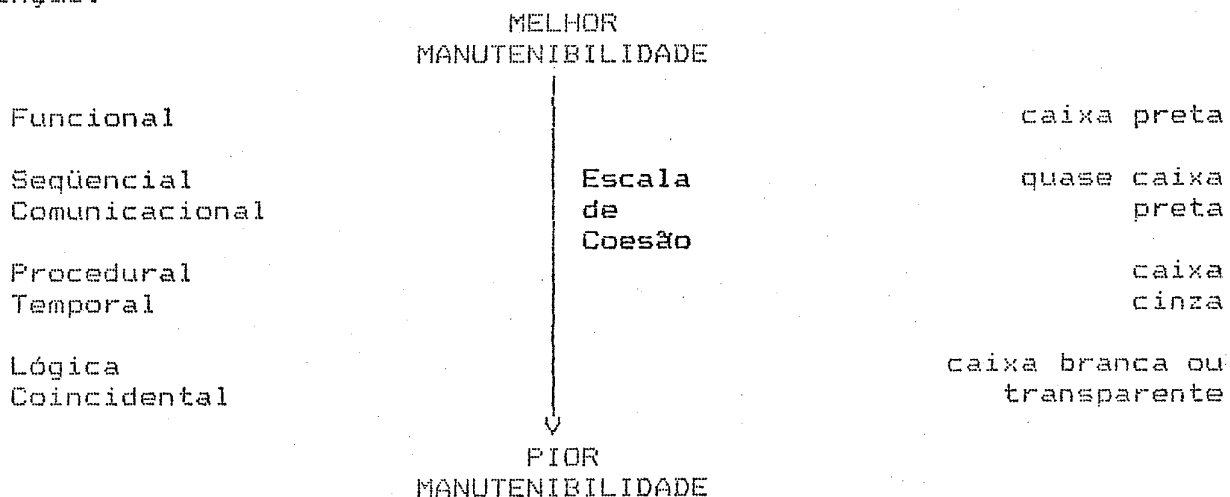
Coesão é a medida da intensidade da associação funcional dos elementos em um módulo. Nesse caso, por elemento deve-se entender uma instrução, ou um grupo de instruções ou uma chamada ("CALL"), para um outro módulo. Ou seja, qualquer parte de codificação que execute alguma ação.

O que queremos são módulos compactos, altamente coesos, cujos elementos são forte e genuinamente relacionados uns aos outros. Por outro lado, os elementos de um módulo não deveriam ser fortemente relacionados com elementos de outros módulos, porque isso levaria a um forte acoplamento entre eles. Já que acoplamento e coesão são ambas maneiras de avaliar modularidade, poder-se-ia esperar que elas fossem interdependentes. Realmente elas são, no sentido de que a coesão de um módulo geralmente determina o quanto ele será acoplado a outros módulos em um sistema.

Coesão é, portanto, a segunda maneira de avaliar se a segmentação de um sistema em módulos foi boa. Na verdade, ter certeza de que todos os módulos têm boa coesão é a melhor forma de minimizar o acoplamento entre módulos. Isso também assegura que as funcionalidades do sistema refletem as funcionalidades do problema original.

A idéia de coesão surgiu com Larry Constantine [4], na década de 60. Ele estava interessado em entender porque as pessoas criavam certos tipos de módulos e em examinar as vantagens e desvantagens relativas a cada um dos tipos. O próprio termo foi emprestado da sociologia, onde significa o relacionamento dos seres humanos dentro de grupos.

Desses primeiros estudos e posteriores aprimoramentos, Stevens[5], Myers[3], Constantine e Yourdon [4][6, pág. 207-32] desenvolveram uma escala de coesão como medida do grau de caixa preta de um módulo e, em função de experiências realizadas, acabou revelando-se uma boa medida da facilidade com que os módulos podem ser alterados nas atividades de manutenção.



2.3.1. Tipos de Coesão

2.3.1.1. Coesão Funcional

Um módulo com coesão funcional contém elementos de dado que contribuem para execução de uma e apenas uma tarefa relacionada ao problema.

Exemplos de módulos funcionalmente coesos são:

- * Calcular o cosseno de um ângulo;
- * Verificar a sintaxe de uma linguagem;
- * Calcular o ponto de impacto de um míssil;
- * Calcular o salário líquido do funcionário.

Verifica-se que cada um desses módulos tem um propósito compacto e único.

Quando o módulo é chamado ele tem apenas uma tarefa para concluir sem ficar envolvido com outras ações que não dizem respeito a essa tarefa. Por exemplo, "Calcular o ponto de impacto de um míssil" deve executar somente isso. Não deve escrever cabeçalho de páginas.

Não importa quão complexo seja o módulo e quantas ações realizam todo seu trabalho. Conseguindo resumi-las em uma "função" relacionada ao problema, então, o módulo é funcionalmente coeso.

Torna-se mais fácil enunciar a regra do seguinte modo: "o exterior (função) de um módulo é mais fácil ser entendido do que seu interior (lógica)" e "um bom módulo é mais fácil de ser usado do que escrito".

No entanto, não se deve pensar que qualquer módulo pode ser considerado como realizando uma única função (ação). Existem vários tipos de módulos que são solicitados pelos seus superiores para a execução de tarefas diferentes que não podem ser reduzidas a uma única função relacionada ao problema.

O fator crucial para decidir o nível de coesão funcional desses módulos não-funcionais é como as diferentes atividades que eles executam estão relacionadas entre si.

2.3.1.2. Coesão Seqüencial

Um módulo seqüencialmente coeso é aquele cujos elementos de dado estão envolvidos em ações tais que os dados de saída de uma ação servem como dados de entrada para a próxima ação. O exemplo da figura 2.9, onde é explicitada a especificação do módulo, ilustra o conceito de módulos seqüencialmente coesos.

Um módulo seqüencialmente coeso geralmente tem bom acoplamento e é de fácil manutenção. Na verdade, é quase tão "manutenível" quanto um módulo com coesão funcional. Entretanto, ele não é tão reutilizável no sistema (ou em outros sistemas) como é um módulo funcionalmente coeso porque ele contém ações que, em geral, não serão utilizadas juntas.

2.3.1.3. Coesão Comunicacional

Um módulo com coesão comunicacional é aquele cujos elementos de dado contribuem para ações que usam a mesma entrada ou a mesma saída. A

figura 2.10 apresenta um caso típico de um módulo comunicacionalmente coeso.

Nesse exemplo, as duas ações de "encontrar nome do cliente" e "encontrar saldo de empréstimo do cliente" utilizam mesma entrada: "Numero de Conta de Cliente".

O acoplamento entre um módulo comunicacionalmente coeso e seu chamador é usualmente aceitável. Tipicamente, ele é estreito de um lado e amplo do outro, dependendo do número de ações que o módulo deve executar.

Os módulos comunicacionalmente coesos são de fácil manutenção, embora possam apresentar problemas.

No exemplo acima, é possível que outro módulo no sistema precise obter o nome do cliente mas não esteja interessado no seu "Saldo de Empréstimo de Cliente". Este módulo deveria descartar os dados do saldo de empréstimo —acoplamento desnecessário— ou precisaria uma codificação própria para acessar o nome do cliente —duplicidade de ação.

Outro problema potencial com módulos de coesão comunicacional é a tentação de partilhar codificações entre ações pertinentes. Isso pode tornar difícil alterar uma ação sem destruir outra.

Por exemplo, o módulo superior na figura 2.11 é comunicacionalmente coeso com a "Tabela de Salários de Funcionários" pois ele tanto gera relatório com os salários de todos os funcionários como calcula a média de salários.

Como ambas as ações precisam acessar todos os salários, um programador poderia ficar tentado a colocá-las na mesma iteração. É possível, no entanto, que, mais tarde, um relatório contendo somente os vinte primeiros funcionários deva ser gerado. Para gerar esse novo relatório, um profissional de manutenção de sistemas precisaria alterar "Calcular Média de Salários", inserindo um teste na iteração para determinar quando o vigésimo funcionário foi processado. No entanto, uma solução melhor seria criar uma iteração separada para cada ação. Mas, nesse caso, não haveria motivo para manter as duas ações no mesmo módulo.

Quase sempre melhora-se as possibilidades de manutenção decompondo-se módulos comunicacionalmente coesos em módulos funcionalmente coesos, como apresentado na figura 2.12.

Módulos com coesão comunicacional e seqüencial se parecem muito pois ambos contém ações organizadas em torno dos dados do problema original. Eles também têm acoplamento bastante claro porque poucos dos seus elementos de dado estão relacionados a elementos de dado em outros módulos.

A principal diferença entre eles é que um módulo seqüencialmente coeso opera como uma linha de montagem, isto é, suas ações individuais devem ser executadas numa ordem específica. Mas, num módulo comunicacionalmente coeso, a ordem de execução não é importante. No módulo descrito no exemplo da figura 2.11, não importa se o relatório de funcionários é gerado antes, depois ou ao mesmo tempo em que a média de salários é calculada.

2.3.1.4. Coesão Procedimental

A medida em que atingimos a coesão procedimental cruzamos as fronteiras dos módulos de fácil manutenção dos níveis mais altos de

coesão para aqueles de manutenção menos fácil e dos níveis médios de coesão.

Um módulo coesivo por procedimento é aquele cujos elementos de dado estão envolvidos em ações diferentes e possivelmente não relacionadas, nas quais o controle flui de uma ação para outra. Vale lembrar que num módulo com coesão seqüencial não é o controle que flui de uma ação para a próxima, mas os dados. A figura 2.13 apresenta um caso de módulo coesivo por procedimento.

Módulos proceduralmente coesos tendem a ser compostos por algumas ações pouco relacionadas entre si, exceto por serem executadas numa certa ordem, numa certa hora. No entanto, essas ações provavelmente estão mais associadas a ações em outros módulos.

É típico de um módulo com coesão procedural que um dado que lhe seja enviado e os dados que ele devolve tenham pouca relação entre si. Também é típico que tais módulos devolvam resultados parciais —por exemplo, um registro lido mas parcialmente editado—, controles, chaves etc..

Coesão procedural poderia ser o tipo de coesão apelidado de "faz-pela-metade", já que esse tipo de módulo tende a fazer apenas parte de uma tarefa.

2.3.1.5. Coesão Temporal

Um módulo com coesão temporal é aquele cujos elementos de dado estão envolvidos em ações que estão relacionadas no tempo.

O exemplo clássico de coesão temporal é um módulo "inicialização", apresentado na figura 2.14.

O módulo "Inicializar" inicializa muitas ações diferentes numa forte trajetória, fazendo com que ele esteja largamente relacionado a vários outros módulos no sistema —condição que é refletida por alto acoplamento com seu chamador.

Módulos com coesão temporal, analogamente àqueles com coesão procedural, tendem a ser compostos de ações parciais cuja única relação é que elas todas acontecem num certo momento. As ações estão geralmente mais relacionadas a ações em outros módulos do que entre si, situação esta que conduz a um elevado acoplamento.

Também são parecidos no sentido de que módulos com coesão procedural ou temporal variam de medíocres a pobres. A diferença entre eles é igual à diferença entre coesão seqüencial e comunicacional: a ordem de execução de ações é mais importante em módulos proceduralmente coesos. Além disso, módulos procedimentais tendem a compartilhar iterações e decisões entre ações, enquanto módulos temporalmente coesos tendem a conter uma codificação mais exata.

Um módulo temporalmente coeso também apresenta algumas dificuldades de um módulo comunicacionalmente coeso.

O programador tem a tendência de compartilhar a codificação entre ações relacionadas unicamente pelo tempo e o módulo se torna de difícil reutilização —tanto nesse sistema como em outros. Obviamente, inicialização e terminação devem ser feitos. Contudo, deve-se evitar o máximo possível construir um módulo coesivamente temporal. Não há regras perfeitas e completas mas elas dizem que, para cada ação, sua inicialização deve ser feita o mais tarde possível e encerrada o mais cedo possível. Outro modo possível de expressar essas regras é

"Inicialize e finalize cada ação tão baixo quanto possível na hierarquia sem introduzir memória de estado".

2.3.1.5.1. Memória de Estado ("State Memory")

Um módulo ordinário "morre" quando retorna o controle de execução a seu superior depois de ter cumprido sua ação, deixando apenas o resultado de seu processamento como legado. Quando o mesmo módulo é acionado novamente para fazer seu trabalho, tudo se passa como se ele estivesse sendo usado pela primeira vez.

O módulo não tem nenhuma memória para guardar acontecimentos anteriores. De fato, como o ser humano, ele não tem a menor idéia se teve ou não uma vida anterior.

Entretanto, existe um tipo de módulo que está ciente do seu passado, através do mecanismo conhecido como "memória de estado" ("state memory"). Memória de estado é um dado interno ao módulo que permanece imutável de uma chamada para outra do módulo. Em algumas linguagens de programação esses dados internos imutáveis são conhecidos como "dados estáticos".

Os profissionais de manutenção de sistemas parecem ter mais problemas em lidar com módulos com memória de estado do que com módulos comuns. Isso porque um defeito associado a um módulo com memória de estado pode ser intermitente (isto é, não sistemático) e depender do estado corrente do módulo. Desse modo, mesmo havendo vários lugares em muitos sistemas onde módulos com memória de estado são inevitáveis, esse mecanismo deve ser evitado. [2, Capítulo 8] apresenta uma regra com a qual pode-se determinar onde haverá necessidade de memória de estado.

2.3.1.6. Coesão Lógica

Um módulo logicamente coeso é aquele cujos elementos contribuem para ações de uma mesma categoria geral, onde a ação ou as ações a ser executadas são selecionadas fora do módulo.

De posse dessa definição, pode-se imaginar o seguinte raciocínio caso esteja planejando uma viagem:

1. Ir de carro;
2. Ir de trem;
3. Ir de navio;
4. Ir de avião.

O que relaciona essas ações? Todas são meios de transporte, certamente. Mas um ponto crucial é que, para qualquer viagem, precisa-se escolher um desses meios de transporte: geralmente, não se usa todos numa viagem só.

Um exemplo completo e interessante desse tipo de coesão é apresentado em [2, Capítulo 7, págs. 140-142].

Um módulo logicamente coeso contém um número de ações com alguma característica comum: para usar o módulo, seleciona-se somente a(s) parte(s) que é(são) necessária(s).

Portanto, um módulo logicamente coeso é uma caixa de surpresas de ações. As ações —apesar de diferentes— são forçadas a compartilhar a única interface do módulo. O significado de cada parâmetro da interface depende de qual ação pretende-se ativar. Para certas ações, alguns dos

parâmetros serão nulos (isto é, não terão influência) —apenas o módulo chamador precisa usá-los e saber seus tipos específicos.

As ações num módulo logicamente coeso devem estar numa mesma categoria, tendo similaridades e também diferenças. Infelizmente, isso induz o programador a juntar a codificação das ações, permitindo que essas ações compartilhem linhas de codificação.

Assim sendo, módulos logicamente coesos não só têm um exterior descaracterizado, com talvez uma dúzia de parâmetros diferentes brigando para usar quatro acessos, mas também o seu interior não tem, em geral, uma boa estruturação. O módulo resultante torna-se de difícil entendimento e manutenção.

2.3.1.7. Coesão Coincidental

Um módulo coincidentalmente coeso é aquele cujos elementos de dado contribuem para ações sem relação significativa entre si como em

1. Arrumar carro;
2. Cozinhar bolo;
3. Levar cachorro para passear;
4. Preencher inscrição para astronauta;
5. Tomar uma cerveja;
6. Levantar da cama;
7. Ir ao cinema.

Um módulo coincidentalmente coeso é similar a um módulo logicamente coeso. Suas ações não estão relacionadas por fluxos de dado ou fluxos de controle. Basta verificar a lista acima para concluir que é impossível executar as sete ações de uma vez. No entanto, as ações num módulo logicamente coeso pertencem, pelo menos, a uma mesma categoria. Em um módulo coincidentalmente coeso, nem isso é verdade.

Analogamente ao tópico referente a "Coesão Lógica", um exemplo completo e interessante desse tipo de coesão é apresentado em [2, Capítulo 7, págs. 143-144].

Coesão coincidental felizmente é rara.

Entre suas causas estão tentativas inúteis para economizar tempo ou memória de computador —ambas, geralmente, sem sucesso— através da incrustação arbitrária de codificação monolítica em módulos. Nesses casos, as alterações necessárias realizadas pelos profissionais de manutenção são consideradas muito difíceis uma vez que os módulos possuem uma coesão medíocre.

Módulos com coesão lógica e coincidental são similares em seus problemas.

Como eles não têm nenhuma função bem definida, o módulo chefe precisa enviar um sinal de controle para dizer a eles o que fazer, isto é, qual ação processar. Isto viola o princípio de caixas pretas, já que o módulo superior precisa conhecer os detalhes internos dos módulos subordinados.

O acoplamento para ambos os tipos de módulos é terrivelmente obscuro.

2.3.2. Determinação da Coesão de Um Módulo

Uma boa maneira é escrever uma pequena frase que, honesta e inteiramente, dá nome ao módulo e descreve sua função. Geralmente, a estrutura da frase revela o nível de coesão do módulo.

Por exemplo:

* Coesão FUNCIONAL

Um módulo executando uma função pode ser resumido por um sujeito, um verbo e um objeto. Por exemplo:

Verbo forte imperativo	Objeto direto singular específico
LEIA	registro de transação de cliente
CALCULE	prêmio de seguro de automóvel
DEDUZA	taxas federais

* Coesão SEQUENCIAL

Várias funções de linha de montagem mostram coesão seqüencial, como em "VERIFICAR CONSISTENCIA DA TRANSAÇÃO E [USA-LA PARA] ATUALIZAR O REGISTRO-MESTRE.

* Coesão COMUNICACIONAL

Várias ações não-sequenciais trabalhando nos mesmos dados são a pista aqui. Por exemplo:

CALCULAR SALARIO MEDIO E MAXIMO DE FUNCIONARIOS

* Coesão PROCEDIMENTAL

Procurar por nomes de procedimentos ou de "fluxos". Por exemplo

ROTINA LOOP ou MÓDULO-CHAVE

ou, em casos piores, nomes com pouco sentido.

* Coesão TEMPORAL

São nomes relacionados com o tempo. Por exemplo

COMEÇO, FIM DE JOB

* Coesão COINCIDENTAL

O nome é pouco significativo. Por exemplo

MODULO-FRUNK, ROTINA DE PROCESSAMENTO

2.3.3. Observações Finais

Existem dois tipos de observação que são considerados na avaliação da coesão existente em módulos: a visão externa e a visão interna.

A visão externa da coesão de um módulo é determinada simplesmente pelo exame de seu nome, supondo que o nome do módulo descreva

precisamente sua funcionalidade e a de seus subordinados.

Um módulo é externamente coesivo se seu nome descreve uma funcionalidade única, utilizando o jargão do sistema que se está desenvolvendo. Uma falha a esse respeito seria um módulo possuindo um nome que indique uma composição de funções —por exemplo, "Atualiza Posição do Cursor & Muda Palavra Chave"— ou um nome que não representa a finalidade específica para a qual o módulo foi criado —por exemplo, "Atualiza Dados".

Deve ser ressaltado que a visão externa da coesão não é um aspecto determinístico da partição de um módulo. Um módulo com muitas funções subordinadas pode, ainda, executar um trabalho simples de um problema específico.

A visão externa da coesão é um bom indicador da compreensão de um módulo, que, naturalmente, conduzirá a uma fácil manutenção.

O outro ponto de vista a ser considerado é a visão interna da coesão de um módulo.

Avaliar a coesão interna de um módulo requer o exame da estrutura interna do módulo, isto é, de sua especificação.

O princípio básico é que módulos cujos elementos de dado são relacionados pela operação sobre dados comuns são mais coesivos que módulos cujos elementos de dado possuem apenas um relacionamento temporal —coesão temporal— e que módulos possuindo os mesmos tipos são mais coesivos que módulos cujos elementos de dado pertençam a alguma outra categoria de classificação.

2.4. COMPLEXIDADE DE MÓDULOS:

Um módulo pode possuir alta coesão e interfaces simples com outros módulos mas, ainda, ser inadequado devido a sua complexidade.

Em geral, um módulo deveria ser suficientemente simples para ser tratado como uma "unidade conceitual" para propósitos de verificação ou modificação.

Não há critério universal que determine a aceitabilidade da complexidade de um módulo. Porém, o tamanho e o número de módulos subordinados são uma boa indicação.

Em relação ao tamanho, um módulo que, quando codificado, produz uma listagem menor do que uma página —ou uma tela— pode ser considerado adequado. Esse critério é baseado na idéia de capacidade de apreensão intelectual das pessoas —principalmente, dos projetistas—, onde ter que mudar de página, ou de tela, reduz a facilidade de entendimento.

Outro aspecto a ser considerado na complexidade de um módulo diz respeito ao seu "fan-out".

2.4.1. Fan-out

O "fan-out" de um módulo é o número de módulos que lhe são imediatamente subordinados. A figura 2.15 ilustra esse conceito.

Um módulo com um fan-out elevado determina maior complexidade lógica. A regra "sete mais ou menos dois" tem sido proposta como limitante superior.

Nesse caso, torna-se um módulo separado uma função contida em um

dados módulo de origem. A essa separação dá-se o nome de fatoramento ("factoring"). No caso de um módulo com excessivo fan-out, o fatoramento deverá definir módulos gerenciadores superiores, subordinados ao módulo anteriormente superior, que controlarão um subconjunto dos módulos originariamente subordinados.

Contudo, esse critério de medição de complexidade deve ser utilizado com um certo cuidado. Pode haver situações em que há um fan-out elevado mas o módulo apresenta uma estrutura lógica simples.

2.5. REUSABILIDADE DE MÓDULOS:

Um aspecto final para refinamento de um diagrama de estrutura hierárquica é a reusabilidade de módulos. Esse aspecto diz respeito a módulos hierarquicamente mais baixos do que a módulos hierarquicamente altos, devido à generalidade destes.

Um instrumento útil de avaliação é o fan-in do módulo.

2.5.1. Fan-in

"Fan-in" é o número de módulos que chamam diretamente o módulo em questão.

É importante a noção de que o "fan-in" de um módulo não deveria resumir-se ao número de chamadores residentes em um mesmo Diagrama Hierárquico de Módulos.

Um módulo, definido inicialmente em um dado diagrama para atender a uma dada tarefa, pode ser projetado para apoiar a execução de outras tarefas, tanto no mesmo sistema (por exemplo, quando este é multiprocessado) como em sistemas distintos (por exemplo, subsistemas alocados a diferentes processadores).

O conceito de reusabilidade do projeto ("design") repousa fortemente na capacidade do projetista de antever essas possibilidades, associadas a uma noção de "fan-in" mais abrangente do que a tradicionalmente empregada.

Módulos reusáveis podem tanto executar ações específicas a uma dada aplicação quanto executar ações para atender a alternativas semânticas distintas.

Deve ser ressaltado que, ao generalizar para criar um módulo reusável, o projetista não deve abandonar os diversos critérios de refinamento de um diagrama de estrutura hierárquica. Produzir um módulo reusável que possui uma interface complexa ou que tem uma baixa coesão não melhorará a qualidade total do diagrama de estrutura hierárquica.

2.6. BIBLIOGRAFIA:

1. "Structured Development for Real-Time Systems"
Volume 1: "Introduction & Tools"
Volume 2: "Essential Modeling Techniques"
Volume 3: "Implementation Modeling Techniques"
Paul T. Ward & Stephen J. Mellor
(Yourdon Press, 1985)
2. "Projeto Estruturado de Sistemas"
Meilir Page-Jones
(McGraw-Hill, 1988)
3. "Reliable Software Through Composite Design"
G. Myers
(New York: Petrocelli/Charter, 1975)
4. "Structured Design: Fundamentals of a Discipline of Computer
Program and Systems Design"
E. N. Yourdon & I. Constantine
(New York: Yourdon Press, 1978)
5. "Using Structured Design"
Wayne P. Stevens
(New York: John Wiley & Sons, 1981)
6. "Classics in Software Engineering"
E. N. Yourdon
(New York: Yourdon Press, 1979)

CONCLUSÃO

Conhecendo os critérios de refinamento mais importantes de um diagrama de estrutura hierárquica, mencionados na seção 2, pode-se concluir os passos 3 e 4, citados na seção 1.

Passo 3: aplicar os critérios de refinamento de diagramas de estrutura hierárquica.

Passo 4: retornar ao diagrama em rede que deu origem ao diagrama de estrutura hierárquica, verificando a consistência e analisando possíveis distorções deste último.

De qualquer modo, há um fator de desenvolvimento não mencionado que deve-se ter em mente em todas as etapas de modelagem: **bom senso**.

O diagrama de estrutura hierárquica constitui um modelo de organização de módulos para um processo ou programa. Especificamente, o diagrama de estrutura hierárquica leva em consideração a natureza seqüencial e hierárquica da maioria das linguagens de programação hoje existentes, assim como tenta realizar a "ponte" entre o esquema de atividades (expresso em "linguagem de rede") e o código. Vale ressaltar que esse diagrama, que descreverá a organização do código, deverá tentar manter-se o mais fiel possível à organização do esquema de atividades, isto é, o projetista deve minimizar distorções que podem incidir nesse processo de transformação. Para isso, são utilizadas os diversos critérios de avaliação e refinamento.

Esperamos que esse trabalho, baseado em literatura recente [1], apresente uma fundamentação conceitual mais sólida do que as até agora apresentadas na literatura e seja eficaz no auxílio aos usuários dos Métodos Estruturados quando tiverem que enfrentar o problema de transformar redes em hierarquias.

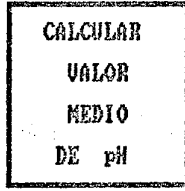


Figura 1.1



Modulo pre-definido

Figura 1.2

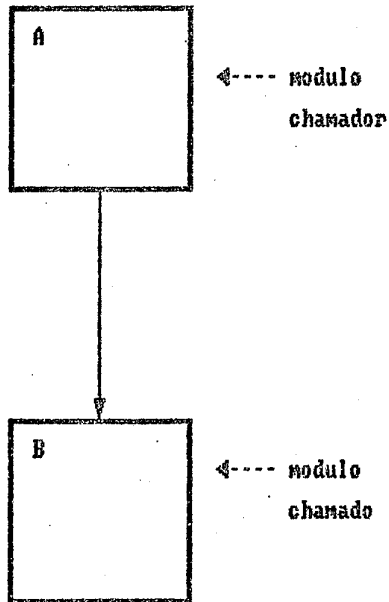


Figura 1.3

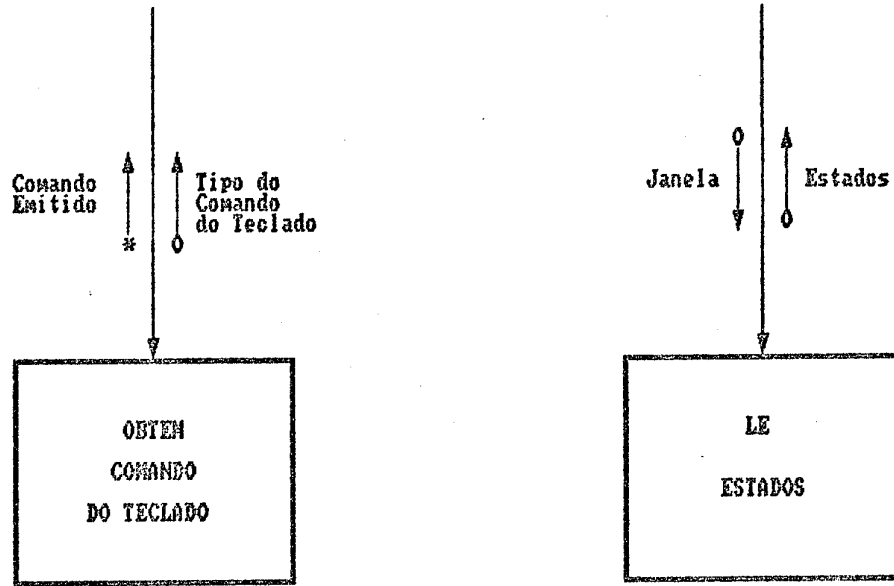


Figura 1.4



Figura 1.5

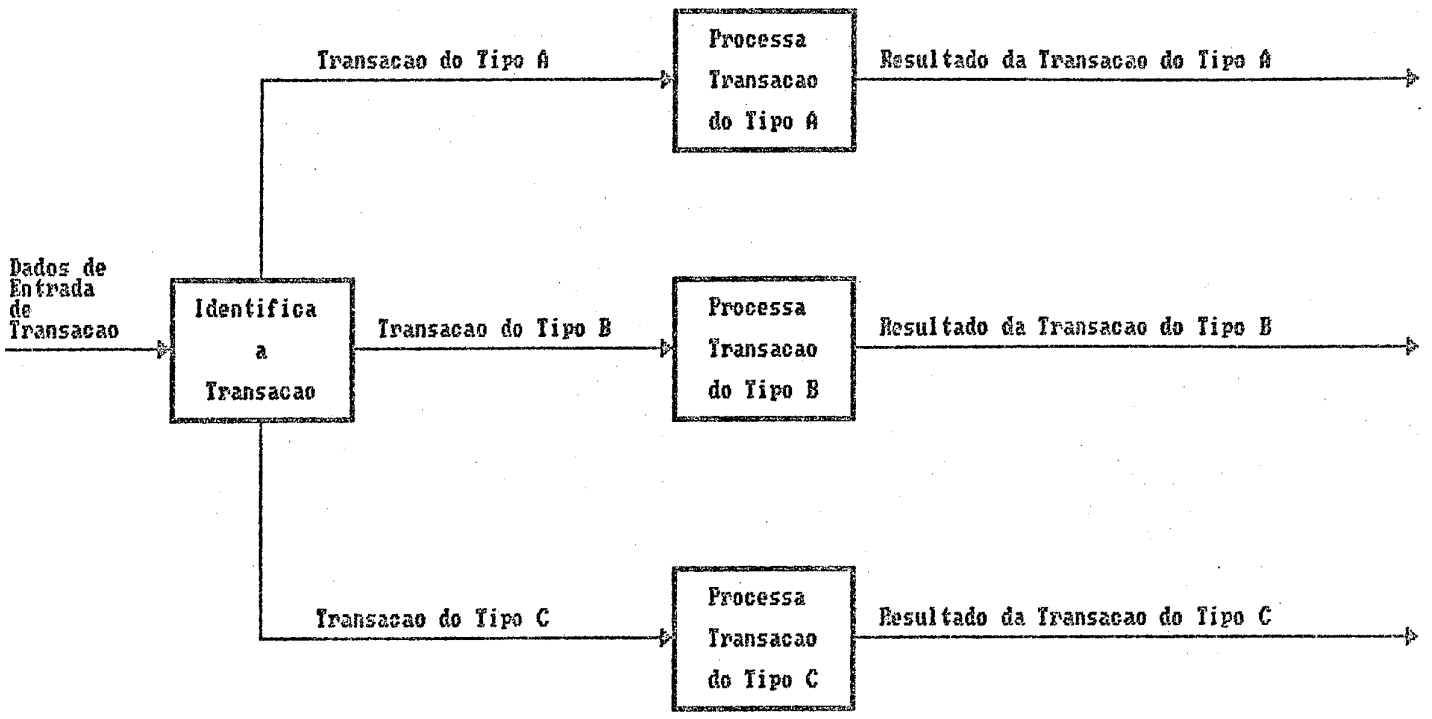


FIGURA 1.6

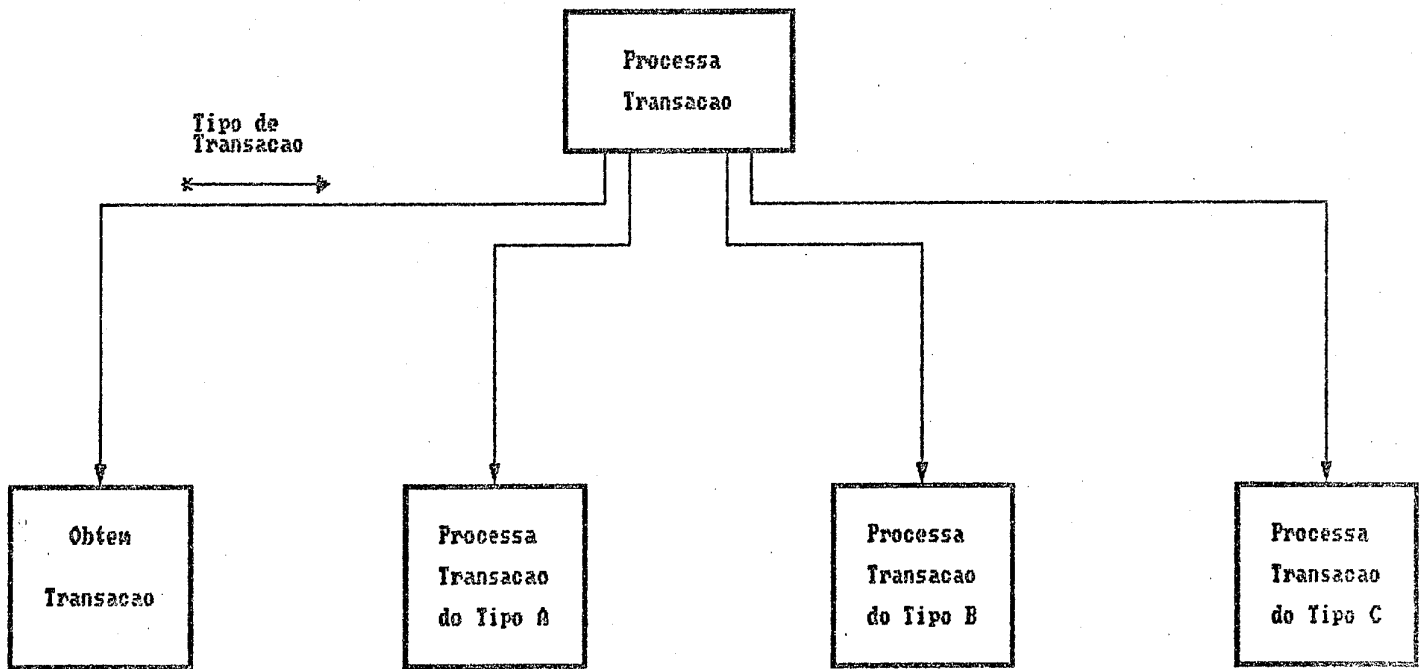


FIGURA 1.7

TABELA 1.1

SINAL DE COMANDO	TIPO DE COMANDO	PROCESSAMENTO REQUERIDO
DIRECIONAR	Direcionar o barco	Direcionar o barco do curso presente para o curso especificado
AJUSTAR	Ajustar o curso do barco	Ajustar o barco ao curso absoluto
DISPARAR	Disparar missil	Missil disparado em direcao especificada
AUTODESTRUICAO	Autodestruicao do barco	Dinamitar barco, apos um tempo especifico

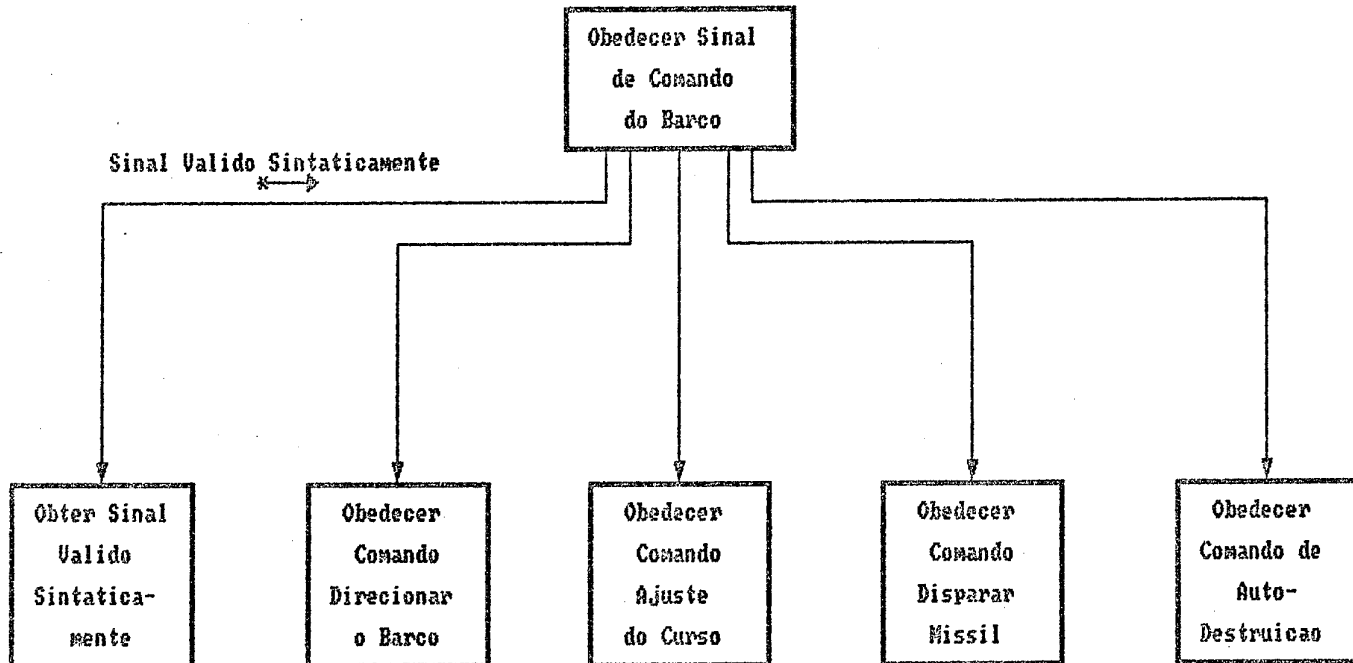
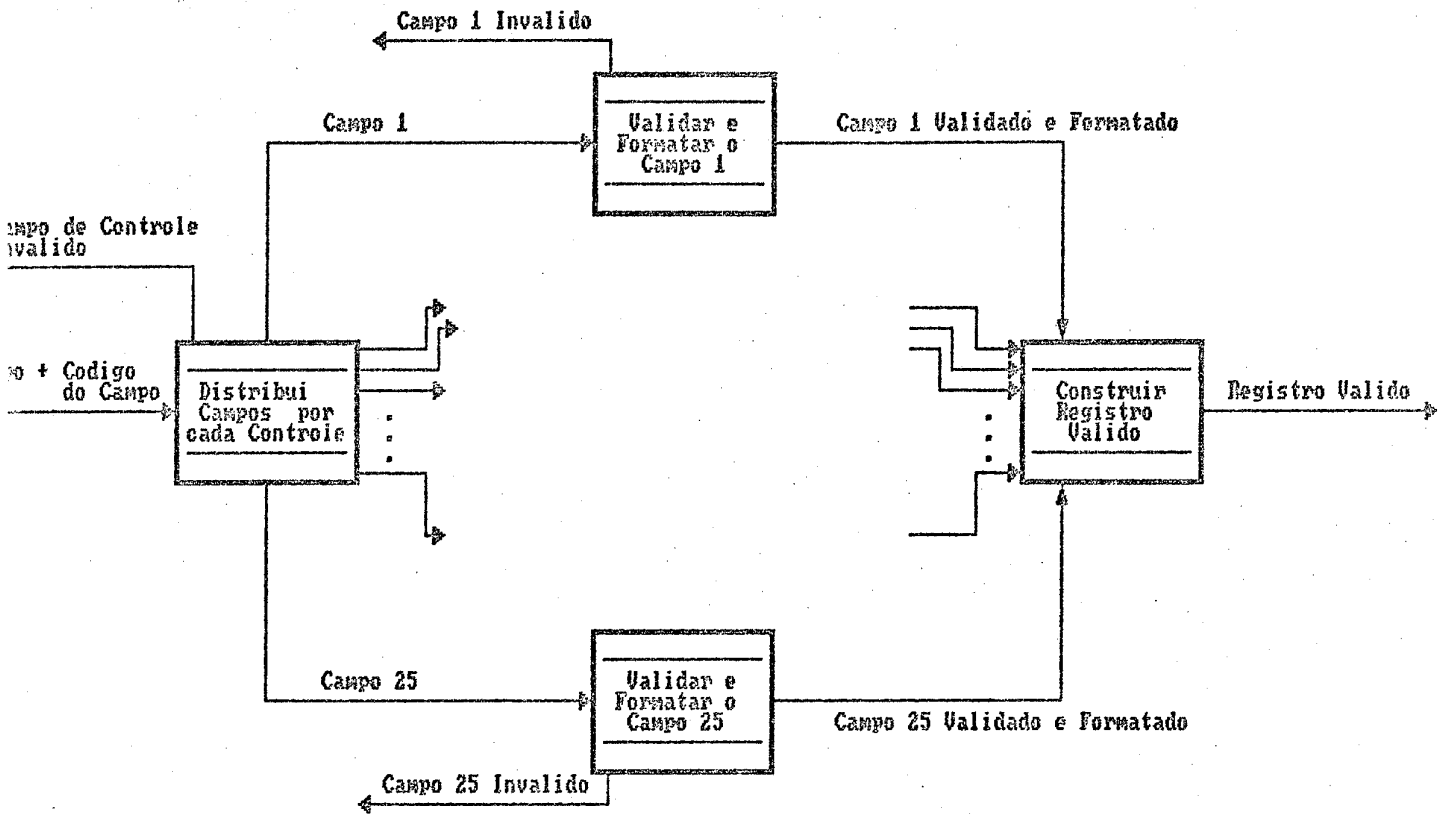
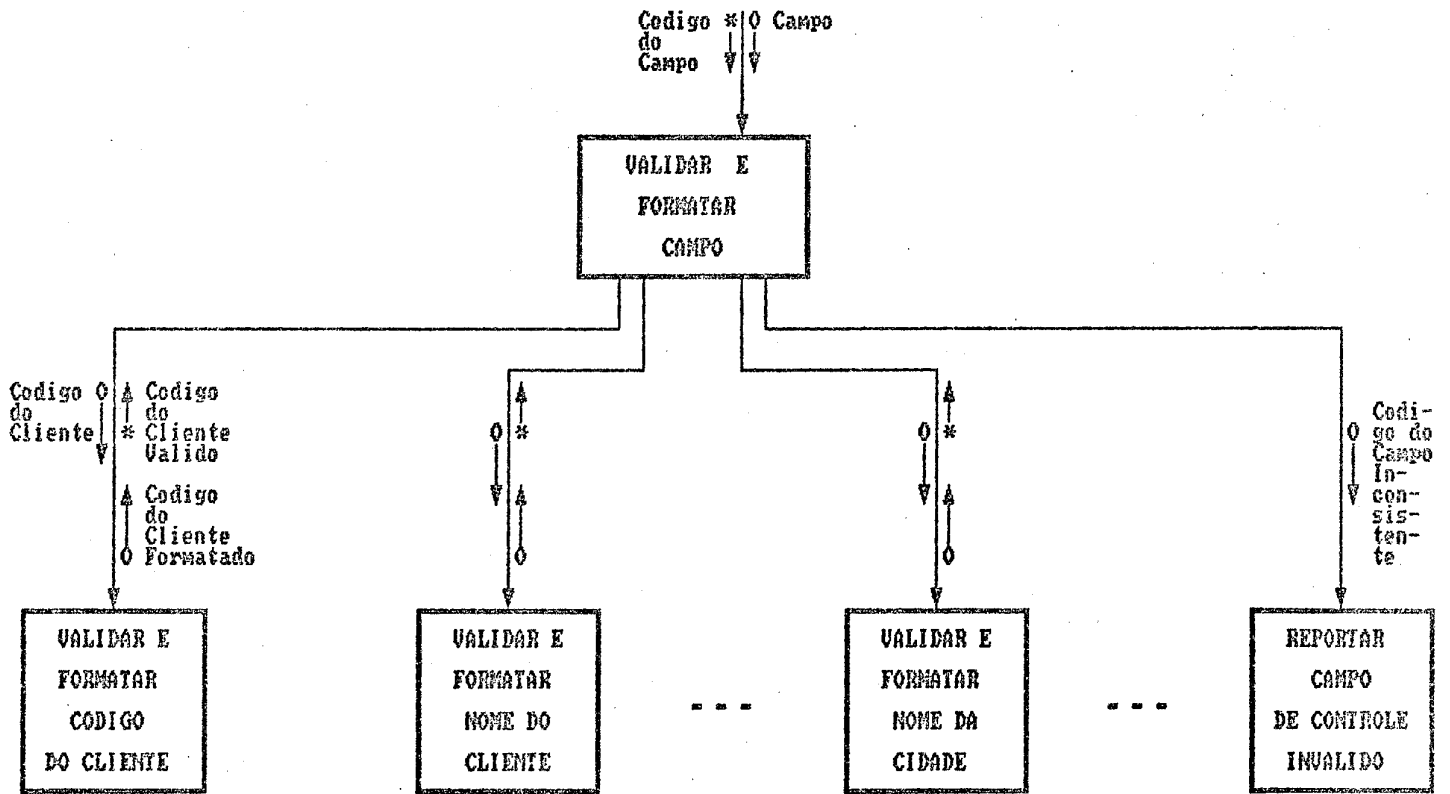


FIGURA 1.8



(a)



(b)

FIGURA 1.9

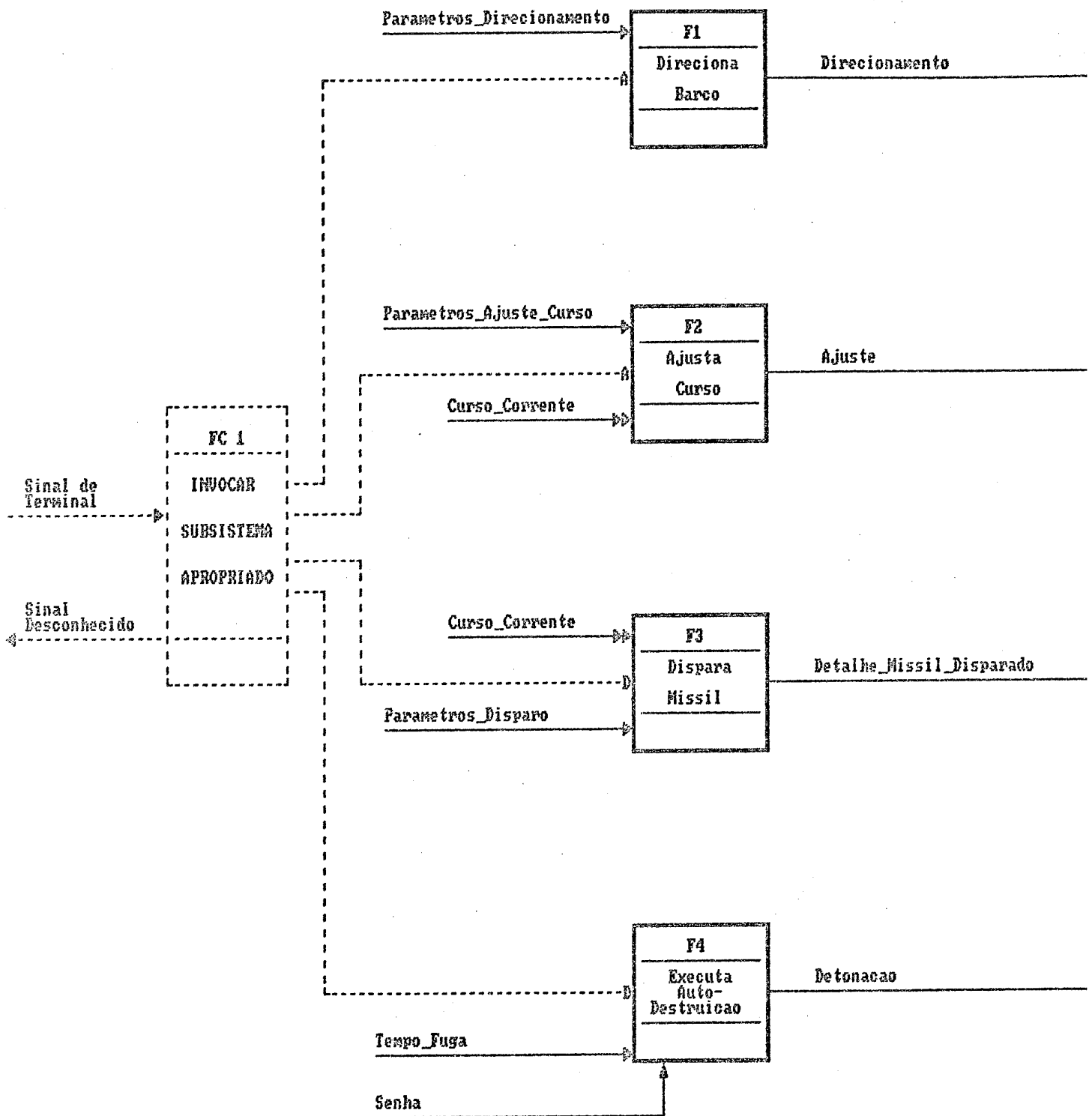
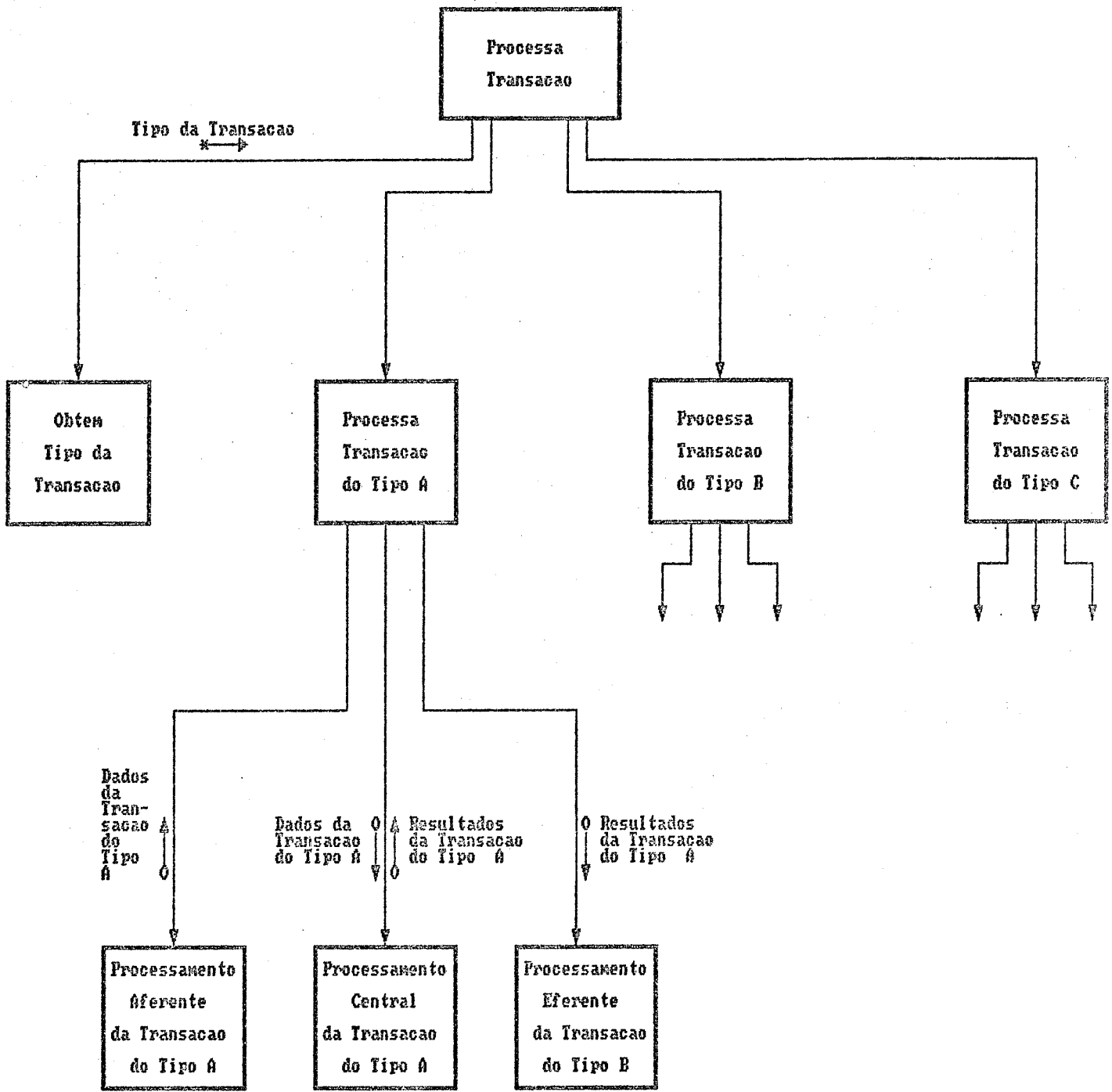
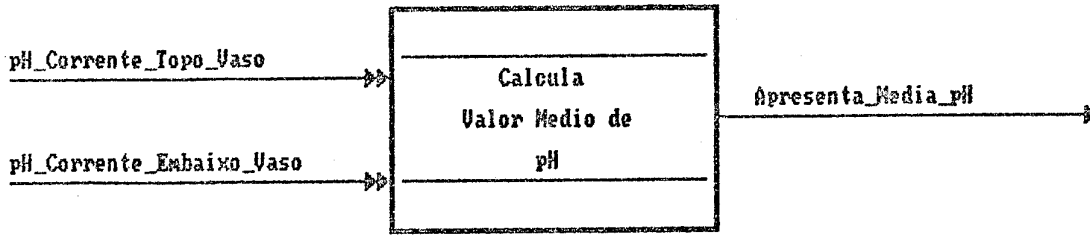


FIGURA 1.10

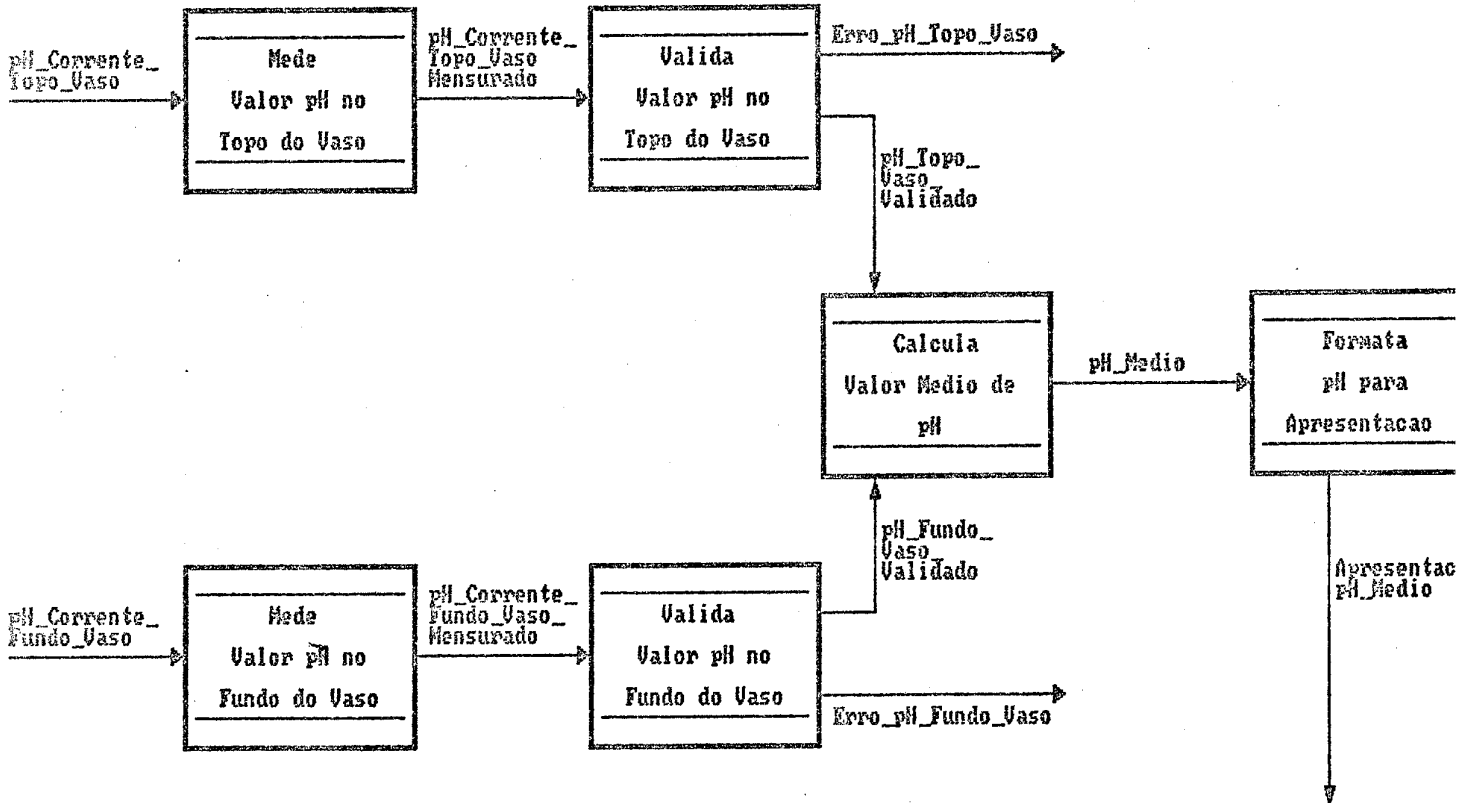


Combinacao de Analise de Transacao e Analise de Transformacao

FIGURA 1.11



Atividade do Modelo da Essencia
(a)



Esquema de Atividades no Modelo da Implementacao

(b)

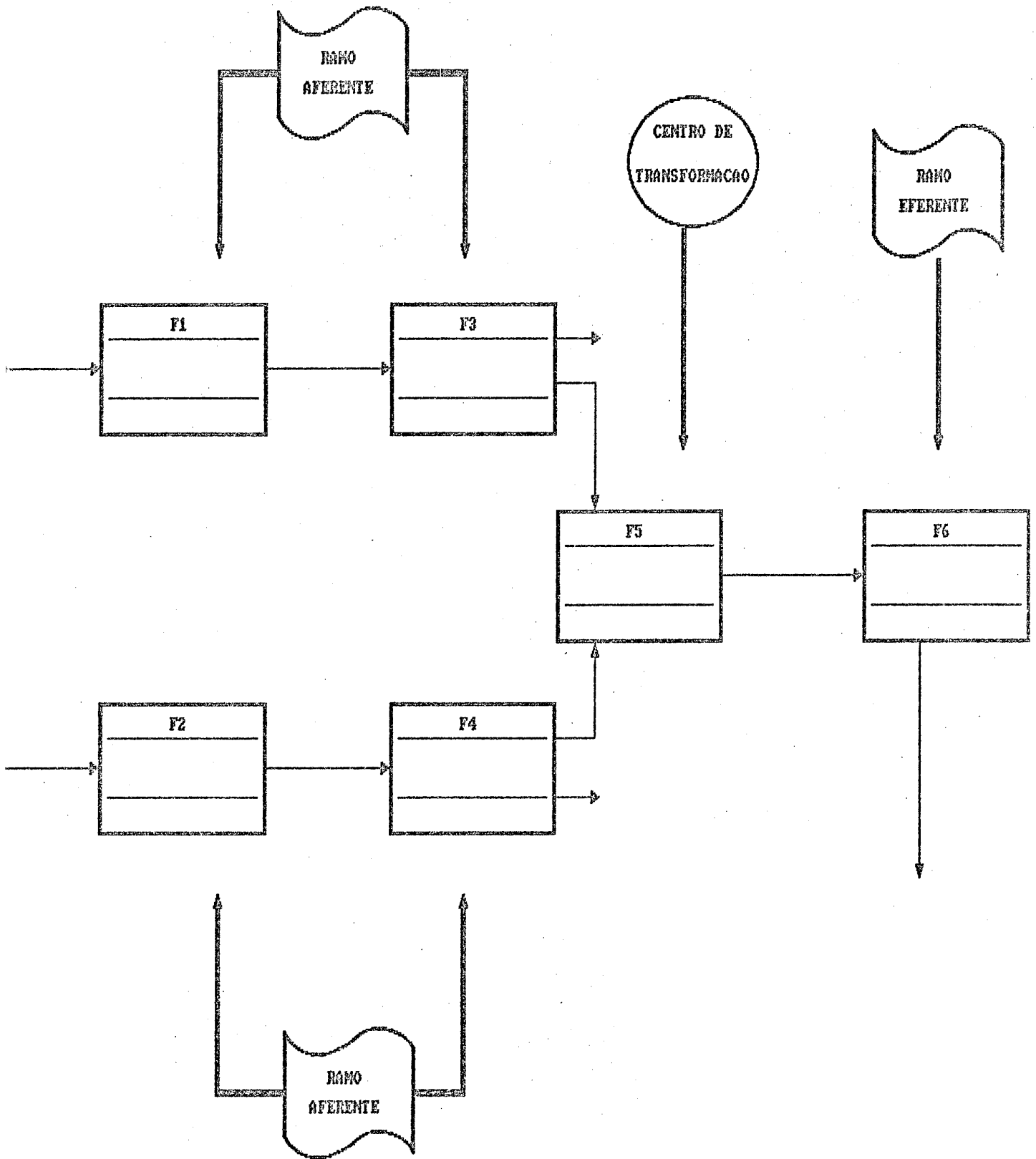


FIGURA 1.13

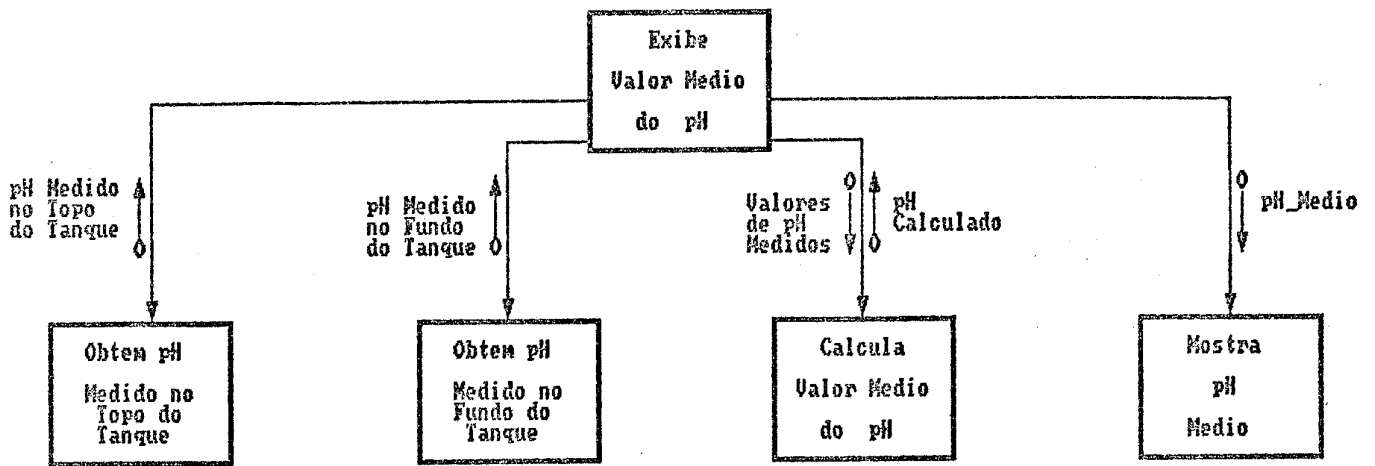


Figura 1.14

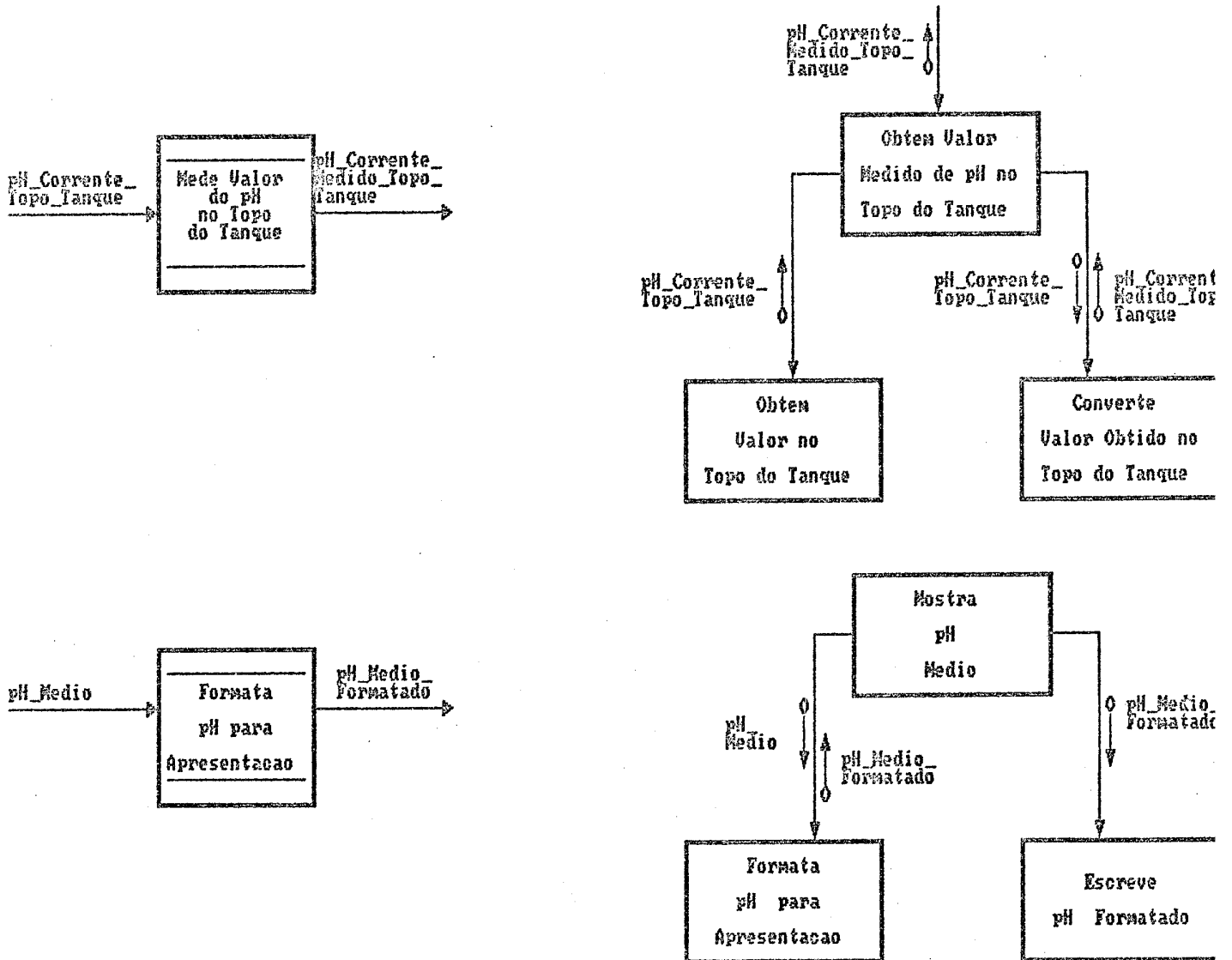


Figura 1.15

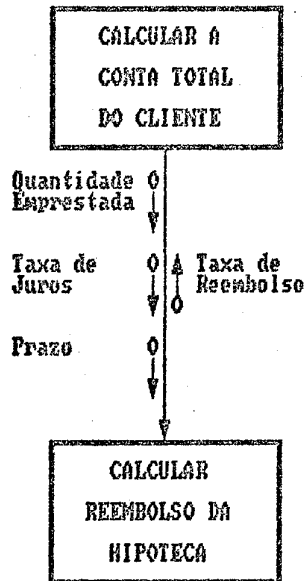


FIGURA 2.1

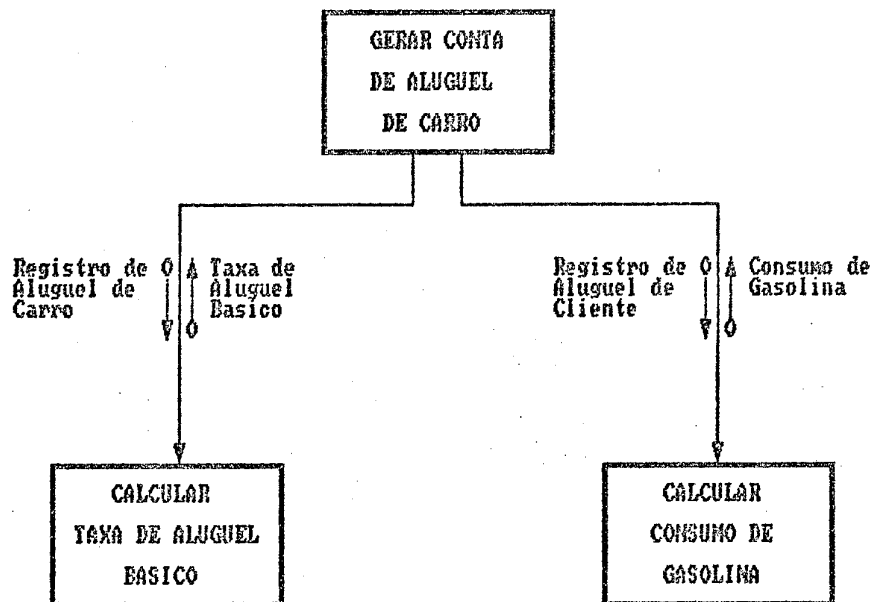


FIGURA 2.2



FIGURA 2.3

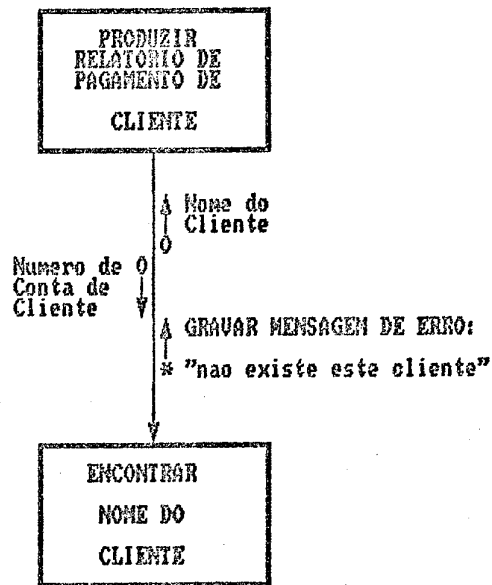


FIGURA 2.4

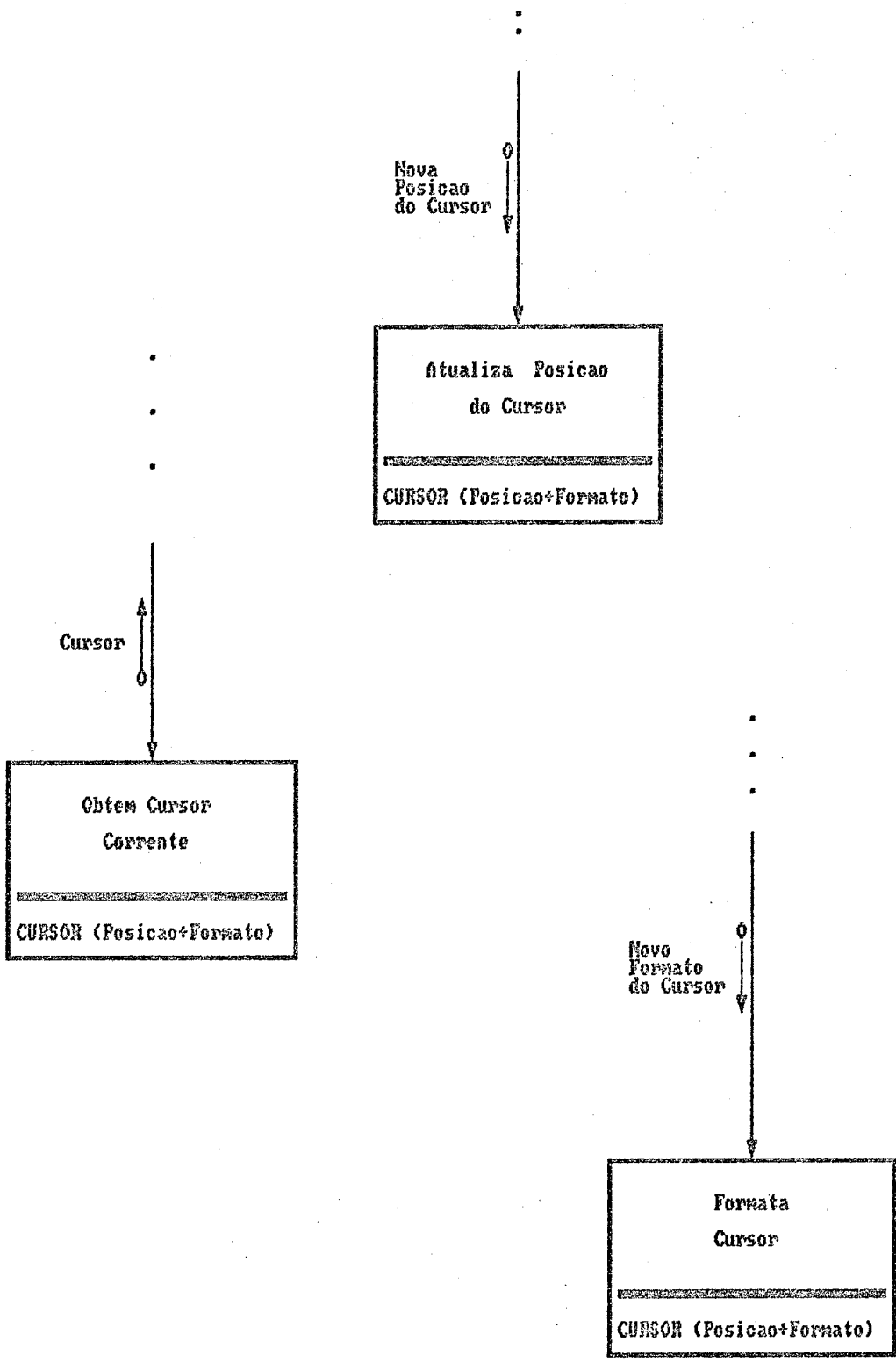
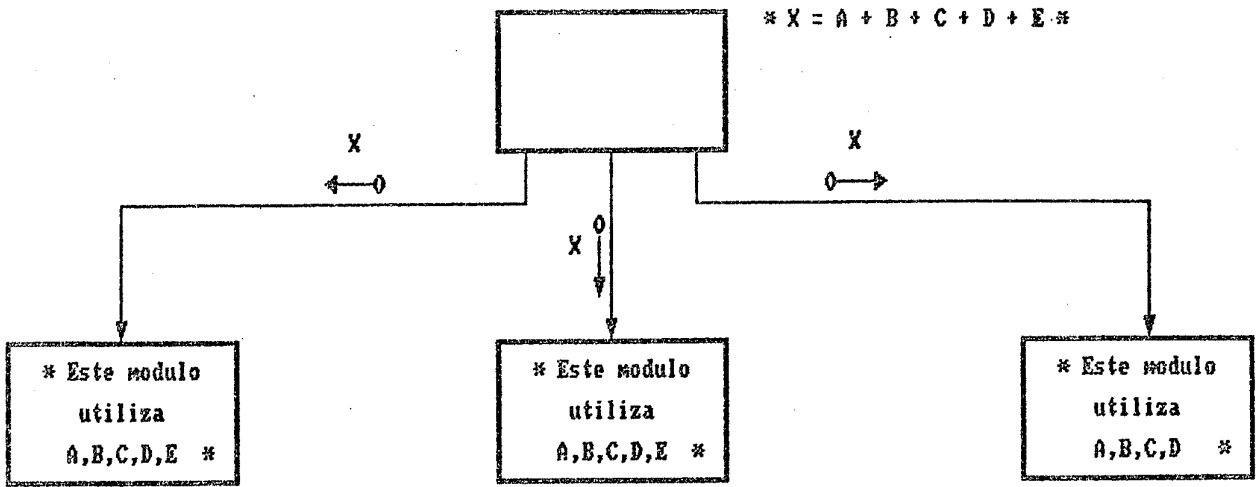
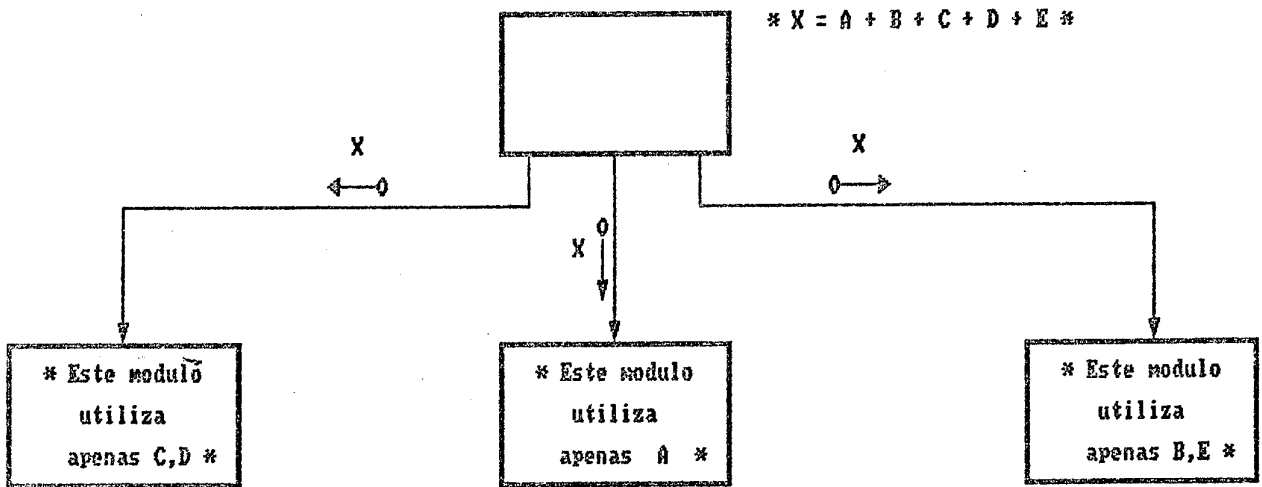


FIGURA 2.5



Uso adecuado de una estructura de datos

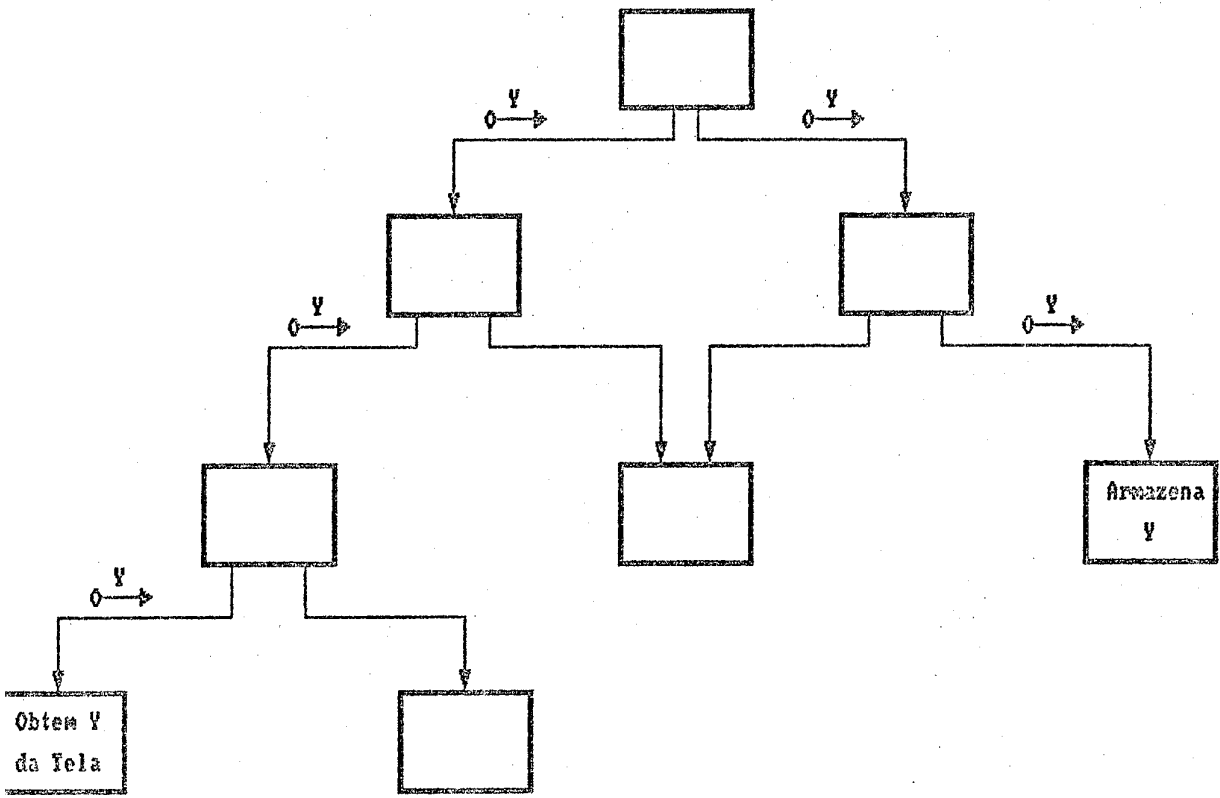
(a)



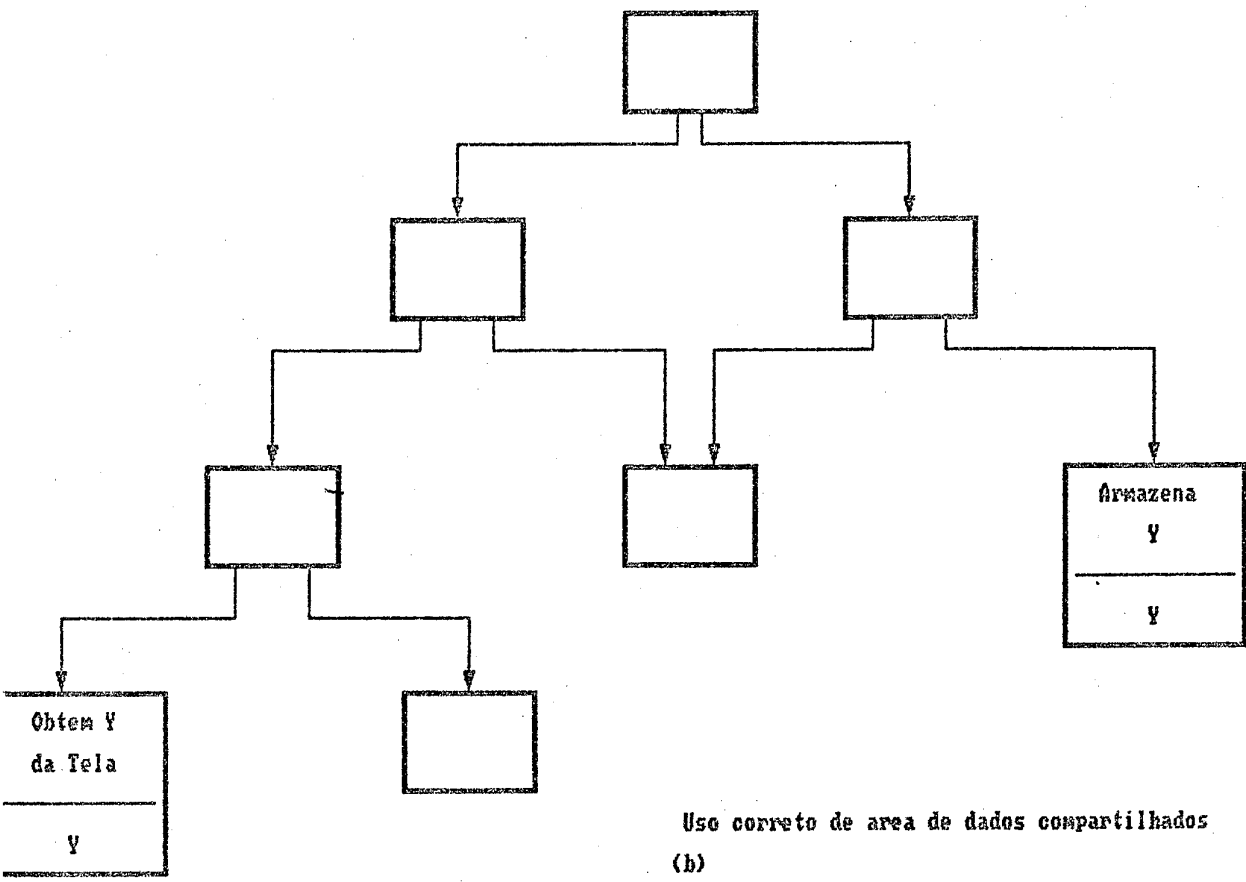
Uso improprio de una estructura de datos

(b)

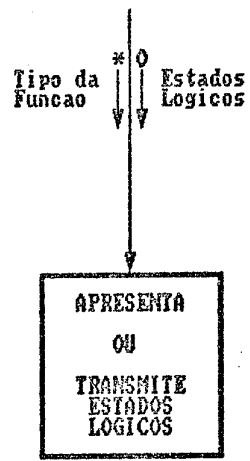
FIGURA 2.6



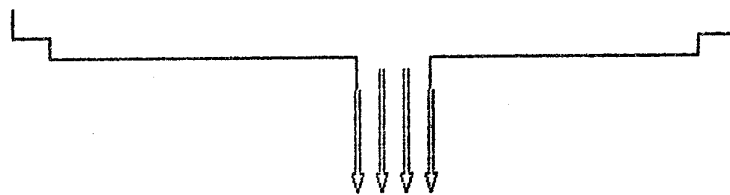
Navegacao de parametros carregada
(a)



Uso correto de area de dados compartilhados
(b)



(a)



(b)

FIGURA 2.8

Modulo Formatar e Verificar Consistencia de Registro

usa registro original
formata registro original
executa cruzamento de campos no registro original
devolve registros consistidos e formatados
fim-do-modulo

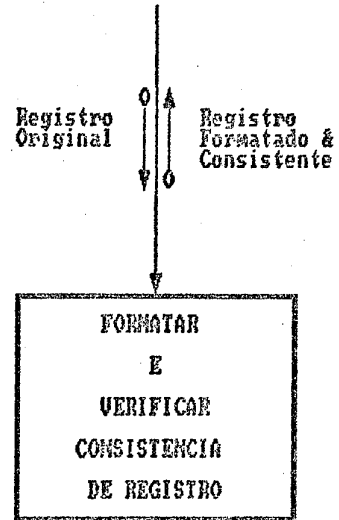


FIGURA 2.9

Modulo Determinar Detalhes de Cliente

usa numero de conta de cliente
encontra o nome do cliente
encontra o saldo de emprestimo de cliente
devolve o nome do cliente, saldo de emprestimo de cliente
fim-do-modulo

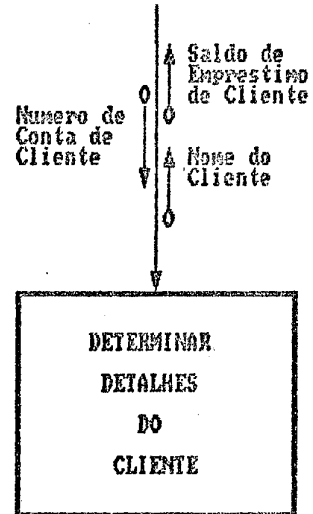


FIGURA 2.10

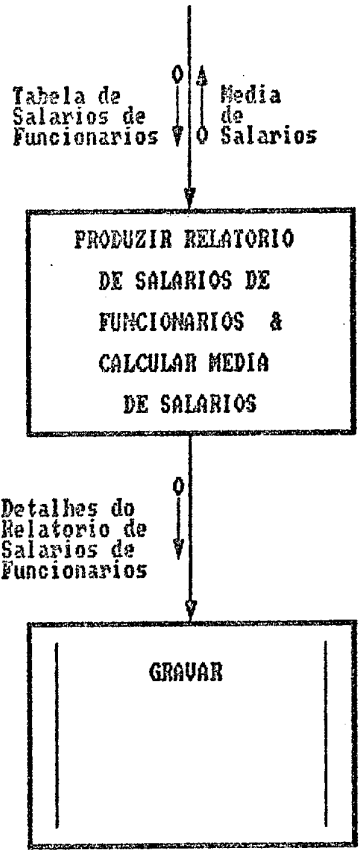


FIGURA 2.11

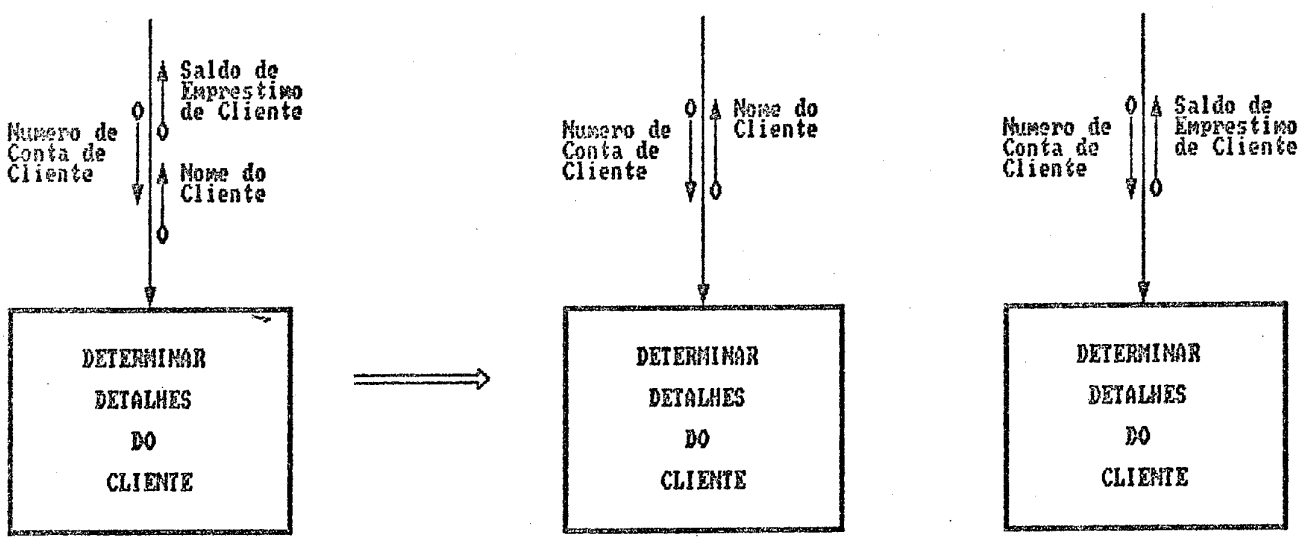


FIGURA 2.12



FIGURA 2.13

Modulo Inicializar

atualizar contador A, contador B, tabela de itens
tabela de totais, chave A, chave B

retroceder fita A

fixar contador A em 0

retroceder fita B

fixar contador B em 0

limpar tabela de itens

limpar tabela de totais

desligar chave A

desligar chave B

fim-de-modulo

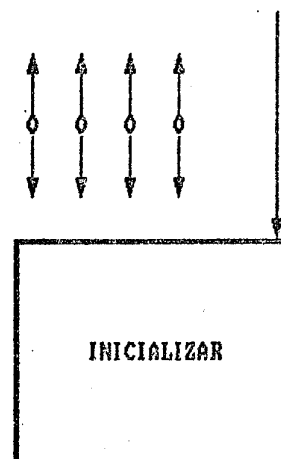


FIGURA 2.14

O modulo tem um fan-out de 3

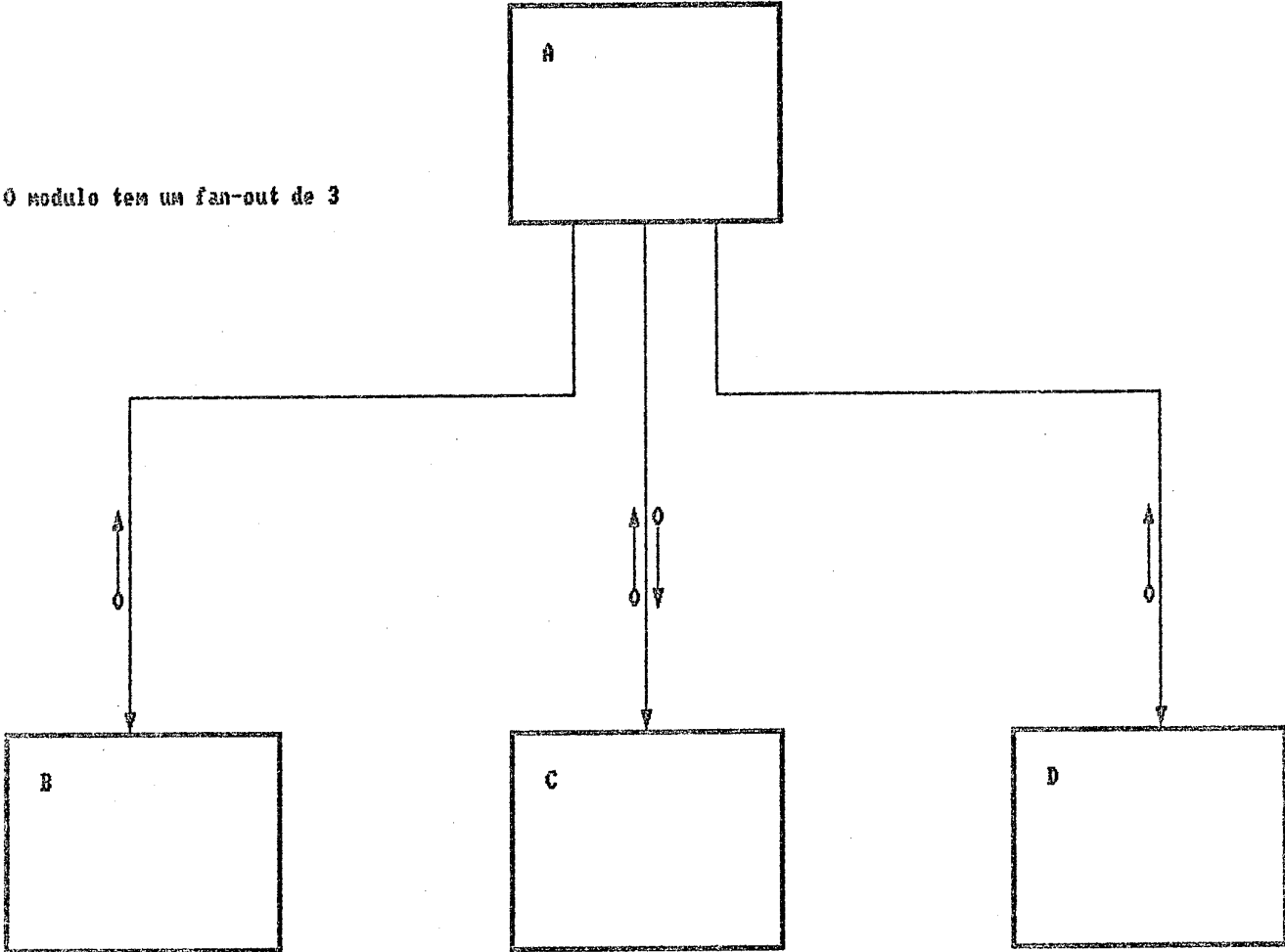


FIGURA 2.15