PUC

ISSN 0103-9741

Monografias em Ciência da Computação
nº 13/95

Sistema de Tempo-Real para Controle de Processos - Design Orientado a Encapsulamento de Dados e a Troca de Mensagens entre Subsistemas Autônomos

Maria Luíza d'Almeida Sanchez
Bruno Maffeo

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA ISSN 0103-9741

Monografias em Ciência da Computação, N° 13/95

Editor: Carlos J. P. Lucena Maio, 1995

Sistema de Tempo-Real para Controle de Processos - Design Orientado a Encapsulamento de Dados e a Troca de Mensagens entre Subsistemas Autônomos *

Maria Luíza d'Almeida Sanchez

Bruno Maffeo

* Trabalho patrocinado pelo Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

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
22453-900 — Rio de Janeiro, RJ
Brasil
Tel. +55-21-529 9386 Telex +55-21-31048 Fax +55-21-511 5645
E-mail: rosane@inf.puc-rio.br

Sumário

1. Introdução 2

2. Problemas do Processo de Desenvolvimento de Sistemas Processo-Controle 4

3. Características de Qualidade 6

4. A Estrutura de Modelagem 7
   4.1. O Modelo da Essência 8
   4.2. O Modelo da Implementação 10
   4.3. O Modelo da Automação 13

5. Modelo da Implementação - Conceitos 13
   5.1. Objetos e Classes 13
   5.2. Sistemas X Serviços 14
   5.3. Interfaces 15
   5.4. Portas 15
   5.5. Interação entre Subsistemas 17
   5.6. Configuração do Sistema 18

6. O Modelo da Implementação 19
   6.1. Requisitos de Concepção de Sistema 19
   6.2. Interface Usuário-Máquina 21
   6.3. Projeto Básico 21
      6.3.1. Derivando Serviços a Partir do Modelo da Essência 22
      6.3.2. Derivando Subsistemas Básicos a Partir do Modelo da Essência 22
         6.3.2.1. Classes x Subsistemas Básicos 22
         6.3.2.2. Identificando Subsistemas Básicos 24
      6.3.3. Identificando a Organização Hierárquica 31
   6.4. Especificando Subsistemas Básicos 32
   6.5. Configuração do Sistema 32

7. Linguagens de Representação 33
   7.1 - Diagramas de Estrutura 33
      7.1.1. Exemplos de Representação 34
      7.1.2. Regras de Consistência 38
   7.2. Diagramas de Comportamento 39
      7.2.1. Exemplos de Representação 40
      7.2.2. Regras de Consistência 48

8. Estrutura do Projeto Básico 48
   8.1. Organização do Projeto Básico 48
   8.2. Descrição Textual Sucinta dos Serviços 49
   8.3. Diagrama de Subsistemas Básicos 50
   8.4. Árvore de Subsistemas 50
   8.5. Dicionário de Dados 50

9. Subsistemas Básicos 51
   9.1. Especificação Detalhada de um Subsistema Básico 51
   9.2. Design de um Subsistema Básico 51

10. Serviços da Interface de Compatibilização 53

10.1. Serviços de Configuração de Software 56
10.2. Serviços de Configuração de Hardware 57
10.3. Serviços de Comunicação 57
10.4. Documentação da Configuração do Sistema 58

11. Emprego do Método 59
11.1. Implantação do Reúso 59
11.2. Prevenção de Deadlocks 60

12. Conclusão 60

Bibliografia 62

Sistemas de Tempo-Real para Controle de Processos - Design Orientado a Encapsulamento de Dados e a Troca de Mensagens entre Subsistemas Autônomos

Maria Luiza d'Almeida Sanchez
UFF- Departamento de Engenharia de Telecomunicações
DI/PUC - Rio
e-mail: mluiza@caa.uff.br
Bruno Maffeo
DI/PUC - Rio
e-mail: maffeo@inf.puc-rio.br

PUC - Rio. Inf. MCC - 013/95

Abstract:
This work presents a method of software design for real-time process-control systems, based on "Information Hiding and Exchange of Messages between Independent Subsystems". Employing the information hiding concept, the method partitions the complex system in more manageable subsystems intended for reuse in other systems that have to perform similar functions. The Incremental Development Strategy is facilitaded, allowing a lower response time by the development team. This strategy permits the production of operational intermediate versions which may be employed by end users and consequently generates a better interaction between users/clients and developers. In general, the final version tends to better satisfy the end user.

Keywords :
Real-Time Systems, Process-Control, Implementation Model, Software Design, Information Hiding, Reuse, Incremental Development, Classes, Objects, Services, Ports.

Resumo:
Neste trabalho, é apresentado um método, baseado em "Encapsulamento de Dados e Troca de Mensagens entre Subsistemas Autônomos", para a geração do "design" (Modelo da Implementação) do software de sistemas de tempo-real para controle de processos. Esse método visa ao domínio da complexidade de sistemas, particionando-os em subsistemas de menor porte, independentes uns dos outros e projetados para reúso, onde o critério mais importante de segmentação do sistema é baseado no encapsulamento de dados. O método facilita o emprego de uma estratégia de Desenvolvimento Incremental, permitindo menor tempo de resposta por parte das equipes de desenvolvimento através da entrega de versões intermediárias operacionais, visando a melhor interação com o usuário/cliente e, em conseqüência, melhor adaptação do sistema a suas reais necessidades.

Palavras Chaves:
Sistemas de Tempo-Real, Controle de Processos, Modelo de Implementação, Design, Encapsulamento de Dados, Reúso, Desenvolvimento Incremental, Classes, Objetos, Serviços, Portas.

1. Introdução

Entre a definição abstrata de um sistema sócio-técnico e sua implementação existe um desnível muito grande, de abstração e de complexidade. O preenchimento da lacuna correspondente constitui, normalmente, uma das etapas mais demoradas do processo de desenvolvimento de sistemas e emprega grande número de recursos materiais e humanos. Nesse sentido, vários esforços vêm sendo realizados visando reduzir custo, prazo e risco associados a projetos de desenvolvimento de sistemas, conforme pode ser encontrado em [Rumbaugh91, Bustard88, Ward84, Jackson83].

A motivação deste trabalho origina-se nesse tipo de questão e seu conteúdo propõe conceitos e técnicas que visam facilitar o design de Sistemas Sócio-Técnicos, em especial sistemas de tempo-real para controle de processos, de grande porte, em ambiente distribuído. Estes últimos apresentam características agravantes quando comparados a sistemas comerciais convencionais, geralmente relacionadas a baixa tolerância referente a requisitos de desempenho e a falhas, a consumo significativo de tempo de desenvolvimento (cerca de 3 a 8 anos) e a emprego de equipes grandes (mais de 15 pessoas). Devido à complexidade dessa classe de sistemas, cada projeto constitui investimento de risco; isto é, existe uma probabilidade alta de não haver sucesso no desenvolvimento e, assim, de inexistir retorno sobre o investimento realizado.

O alto custo do desenvolvimento e o tempo excessivo decorrido entre a elicitação de requisitos e a entrega do produto é muito grande geram insegurança e insatisfação no cliente/usuário que não consegue visualizar o retorno do alto investimento que está realizando. Esse grau de insatisfação pode ser tão intenso que provoque o cancelamento do projeto.

O longo tempo de desenvolvimento do software, acoplado à rápida evolução do hardware disponível no mercado, acarreta muitas vezes o uso de hardware que já está obsoleto no momento em que o software é dado como pronto.

A eventual necessidade de desenvolver simultanemente hardware e software implica geralmente teste do software empregando hardware ainda não confiável. Isso introduz outras variáveis no período de teste de software, com repercursão negativa na capacidade de identificação de fontes de erro.

Equipes numerosas durante o período de desenvolvimento sofrem mudanças e a integração de um outro membro na equipe é geralmente um processo doloroso pois seu treinamento é feito sobre uma documentação massiva e pouco modular. Essas equipes necessitam também de procedimentos que garantam uma troca eficiente de informações para acompanhar a evolução do desenvolvimento e das mudanças que necessitem ser introduzidas.

Durante a construção e a manutenção de um sistema de computação, um dos maiores problemas é o gerenciamento de mudanças. Essas podem ser fruto de erros cometidos pela equipe durante o processo de desenvolvimento bem como devido a novos requisitos do cliente/usuário. Trata-se de sistemas que devem evoluir conforme a evolução das necessidades operacionais, tecnológicas e do ambiente externo e a introdução de um sistema de software é um estímulo à mudança. Geralmente, correção de erros e introdução de novos requisitos implicam alterações de design. Mudanças podem requerer modificações simples, extensões de funções já existentes ou introdução de novas funções. As chamadas mudanças evolucionárias são as mais difíceis de implementar. Os sistemas de tempo-real, devido a sua complexidade e, portanto, alto custo e tempo de desenvolvimento, devem possuir uma vida útil grande (comparada ao tempo de desenvolvimento) e, para tal, ser projetados para acomodar mudanças de forma flexível.

As dificuldades mencionadas não ocorrem apenas no desenvolvimento de sistemas de tempo-real para controle de processos. No entanto, exigências específicas, tais como concorrência, ambiente distribuído, rígidos requisitos de desempenho e tolerância a falhas, aumentam a complexidade do sistema a desenvolver e agravam os problemas do processo de desenvolvimento.

Na Engenharia de Software, o domínio da complexidade do processo de desenvolvimento de um sistema se dá através de uma sistemática de segmentação do processo de desenvolvimento em etapas com resultados bem definidos. Durante cada etapa, o domínio da complexidade se dá através da segmentação do sistema em subsistemas de acordo com a abordagem característica dessa etapa. Esse conjunto de componentes - subsistemas - que trabalham cooperativamente para um propósito comum pode, tecnicamente, ser descrito como uma relação entre entradas (presentes e passadas), saídas e tempo.

Nessas condições, é fundamental empregar ferramentas conceituais e técnicas apropriadas, que permitam a redução dos custos e do risco durante a construção do sistema e do custo de sua manutenção evolutiva e/ou corretiva, que apoiem o estudo da concorrência inerente ao software de controle e que aumentem sua portatilidade para diversas arquiteturas de hardware. Isso viabilizaria projetos importantes que hoje não são executados pois o cliente/usuário não se dispõe a enfrentar todos os ônus que podem dele decorrer.

O método de design aqui apresentado utiliza técnicas de abstração que permitem reduzir a complexidade, escondendo informações ou agregando informações similares ("information hiding") [Parnas , NRL , Gomma93]. Ele também determina a forma a ser empregada pelos projetistas para registrar as decisões, facilitando a manutenção futura do sistema. Esse método favorece o tratamento dos problemas anteriormente apresentados, minimizando-os através do atendimento a requisitos que o processo de design do sistema deva satisfazer. São eles:

- Garantia de Qualidade Assegurada por Construção
- Redução de Custos (cumprimento de orçamentos planejados)
- Redução de Tempo Gasto na Implementação (cumprimento de cronogramas planejados)
- Otimização da Gerência das Equipes Envolvidas
- Redução do Impacto Devido a Mudanças nas Equipes de Desenvolvimento

As seções 2 e 3 deste trabalho ressaltam os problemas encontrados no decorrer do desenvolvimento de sistemas processo-controle em tempo-real e as características de qualidade a serem exigidas de produtos desse processo de desenvolvimento visando minimizar os efeitos desses problemas.

A seção 4 descreve em linhas gerais a Estrutura de Modelagem proposta para a obtenção de produtos com a qualidade desejada. As seções 5 e 6 abordam em detalhe o Modelo da Implementação (conceitos nos quais se baseia e o processo de modelagem). A seção 7 introduz a linguagem de representação para o modelo da implementação (sintaxe e semântica). As seções 8 e 9 tratam, respectivamente, a estrutura de documentação, relativa ao projeto básico e aos subsistemas básicos, correspondente ao design do software do sistema.

A seção 10 descreve os serviços a serem prestados por uma Interface de Compatibilização à aplicação, especificados durante o design. Essa interface é um dos principais responsáveis pela obtenção das metas de qualidade fixadas para o software. A seção 11 discute vantagens e devantagens do método em relação ao reúso e à prevenção de deadlocks e, finalmente, a seção 12 conclui este trabalho.

2. Problemas do Processo de Desenvolvimento de Sistemas Processo-Controle

O objetivo básico de um sistema processo-controle é manter alguma propriedade do processo a um valor específico no decorrer do tempo. A porção de controle do sistema depende das condições a serem satisfeitas pelo processo [Levenson90]. Os subsistemas digitais de controle interagem com processos físicos e a modelagem do ambiente externo é implicitamente refletida na modelagem do software. Acopladas a esse objetivo existem restrições a suas condições de operação [Levenson90] originárias de:

- Considerações de Qualidade;
- Limitações Físicas e Capacidades do Equipamento;
- Características do Processo;
- Considerações de Segurança.

Essas restrições são traduzidas em rígidos requisitos - de desempenho, tolerância a falhas e outros - que limitam o espaço de soluções de uma implementação aceitável para o sistema processo-controle como um todo e, em particular, para a fração sob a responsabilidade do subsistema digital de controle, no que diz respeito ao hardware e ao software. Esse subsistema de tempo-real possui características técnicas importantes que influenciam seu processo de desenvolvimento. São elas:

- o processamento deve ter elevada disponibilidade confiável e satisfazer continuamente requisitos de tempo;
- a execução ocorre em ambiente externo imperfeito;
- a segurança na presença de falhas de software e hardware deve ser assegurada.

Ao examinar sistemas complexos, Perrow [Perrow84] conclui que acidentes não podem ser completamente eliminados. Eles decorrem da complexidade e do acoplamento existente entre os elementos que compõem o sistema. Quanto maior a complexidade do sistema maior a probabilidade de ocorrência desses ditos "acidentes normais". Computadores aumentam a complexidade dos sistemas [Levenson90] e, em conseqüência, a probabilidade de acidentes. Esse aumento de complexidade se dá sobretudo devido à enorme flexibilidade do software, a qual permite a construção de sistemas processo-controle mais complexos e a introdução de novos tipos de controle também mais complexos [Levenson90]. O acoplamento do sistema também aumenta pois, por si só, o computador é um fator de acoplamento. Mas, a introdução de sistemas de computação apresentam vantagens sobre os controles convencionais (mecânicos, analógicos e humanos), o que os leva a ser cada vez mais utilizados.

Nessas condições, os modelos a serem empregados no processo de desenvolvimento não podem focalizar apenas a validade da implementação em relação a uma especificação de requisitos. Para garantir segurança, disponibilidade, confiabilidade e desempenho, os modelos empregados em todas as fases do desenvolvimento, em especial durante a elicitação de requisitos, devem especificar e prever o comportamento (normal e anormal) do ambiente externo sob todas as circunstâncias, explicitando as restrições do processo e incluindo o fato de que a introdução de um sistema de computação pode causar mudanças nesse comportamento.

Para atender a requisitos de tolerância a falhas de hardware e software do sistema de controle, fundamental para a confiabilidade/disponibilidade do sistema, os subsistemas digitais de controle são normalmente implementados em ambiente distribuído com processadores locais alocados a sensores/atuadores. Sistemas distribuídos agregam complexidade pelo fato de atribuírem funcionalidade a componentes computacionais (atribuição de uma fração do software a um

elemento do hardware). Isso aumenta significativamente a complexidade da interface necessária para estabelecer a cooperação entre os diversos componentes de software. Assim sendo, questões associadas à confiabilidade devido à perda de comunicação entre processadores precisam ser consideradas muito mais criticamente do que em sistemas convencionais.

Critérios de segmentação de sistemas de processo-controle devem atender à necessidade de manter simplicidade, minimizando a interconexão entre subsistemas e privilegiando a comunicação através de dados processados ao invés de dados brutos. Questões de alto nível de abstração, tais como a seleção de arquiteturas de hardware e software e a garantia de desempenho e confiabilidade do sistema computacional integrado, devem ser analisadas antes do projeto individual de componentes.

Tais sistemas possuem demanda de processamento em parte cíclica e em parte como resposta a eventos externos com prioridades diferentes de atendimento. A resposta a esses eventos submete-se a restrições temporais impostas pelo ambiente externo. Isso implica que subsistemas de controle em tempo-real são, geralmente, Sistemas Concorrentes. Um **Sistema Concorrente** caracteriza-se por conter componentes cooperativos - **Processos** -, com execução intercalada no tempo ou até simultânea, caso o computador disponha de mais de um processador. Para que processos cooperativos, executando a velocidades diferentes, possam interagir em determinados pontos de execução, devem existir mecanismos de sincronização/comunicação entre eles. A cooperação entre componentes se dá através de troca de mensagens ou compartilhamento de dados com exclusão mútua. Isso aumenta o grau de risco do desenvolvimento, a exigência de qualidade e o custo.

Eventos críticos, muitos deles correspondentes a falhas ocorridas no ambiente externo, devem associar-se a ações imediatas do subsistema de controle do sistema processo-controle visando garantir a segurança do processo. A eficiência e a eficácia desse subsistema são condições necessárias para seu emprego.

Garantir em tempo de Design que o sistema alcançará os requisitos de desempenho necessários não é tarefa simples. No entanto, um sistema de tempo-real só serve se cumprir os requisitos de desempenho especificados pelo cliente/usuário. Modelando a concorrência/seqüencialização existente entre atividades do sistema e considerando-se o pior caso (carga máxima de processamento) é possível estimar o tempo máximo no processamento de um evento. Esse dado, aliado a técnicas de Planejamento de Capacidades [Gomma93, Menascé85], permite prever, aproximadamente, o desempenho final da tecnologia de implementação proposta. Entretanto, tratando-se de uma aproximação, os resultados obtidos após a automação podem não ser satisfatórios. Mesmo que o desempenho do sistema esteja satisfatório, mudanças precisam ser implementadas em qualquer sistema e a introdução de uma função a mais pode ser o suficiente para invalidar a implementação alterada. Daí a necessidade de uma técnica de design que permita mudanças nas arquiteturas de hardware e software, compatíveis com os requisitos de desempenho, durante e após a automação do sistema. Flexibilidade em relação a mudanças na arquitetura de software (número de processos, alocação de processo a processador) e portatilidade em relação a diversas arquiteturas de harware (número de processadores, características da rede de comunicacão de dados) e de software básico (sistema operacional incluindo tratamento de comunicações em rede) são exigências difíceis de satisfazer através do uso das técnicas de design hoje existentes para desenvolvimento de sistemas de tempo-real.

Atender a requisitos de tolerância a falhas (atendendendo integralmente à funcionalidade do sistema ou parcialmente quando em execução degradada) não introduz nova funcionalidade no sistema mas exige a definição de arquiteturas de hardware e software que, em caso de falha de um processador, fique garantida total ou parcialmente a função a ele alocada. A introdução dessa

capacidade, por exemplo através do uso de um processador em "hot stand by", não deve aumentar a complexidade do design do sistema, apenas refletindo-se em níveis mais baixos de abstração onde estarão definidos os processos do sistema e sua alocação a processadores - configuração de hardware e software.

3. Características de Qualidade

Quatro métricas foram consideradas relevantes na avaliação da qualidade de sistemas: Modularidade, Manutenibilidade, Portatilidade e Reúso.

A **Modularidade** reflete o baixo acoplamento entre subsistemas componentes e assegura a capacidade de alteração em um dos subsistemas com o mínimo de impacto para os demais subsistemas. Alta Modularidade aumenta a clareza do design e facilita a implementação, teste, documentação e manutenção. Ela possibilita o acréscimo, modificação ou substituição de funções sem influências significativas nas demais funções do sistema.

A **Manutenibilidade** reflete o grau de facilidade com que correções ou novas funções são incorporadas ao sistema ou a um subsistema, visando mantê-lo operacional, eficiente e eficaz. Ela pode ser expressa em termos de outras medidas de qualidade de sistemas, tais como: simplicidade das soluções, clareza das decisões, Modularidade e boa documentação (registro dos modelos e das decisões de design). Alto grau de Manutenibilidade facilita o entendimento do sistema por pessoas externas à equipe de desenvolvimento e, conseqüentemente, reduz problemas decorrentes de mudanças na equipe, além de permitir maior agilidade na correção de erros.

A **Portatilidade** reflete o poder de transferência do software para outro computador ou ambiente de software básico. Alto grau de portatilidade permite a migração do software, com pequenas alterações, para uma nova Arquitetura de Hardware, com o objetivo de melhor desempenho ou manutenção mais barata.

O **Reúso** reflete o quanto um subsistema pode ser utilizado em outros sistemas. O desenvolvimento de um subsistema reusável requer esforço maior do que o de um não reusável. Seu reúso em outro sistema também implica esforço adicional para compreender sua especificação e decisões de design. O emprego de subsistemas reusáveis promete reduzir o tempo de desenvolvimento e aumentar a qualidade, o que proporcionaria também redução de custos [Tracz94]. Na maioria dos casos isto ainda está por se materializar, no entanto existem exemplos de sucesso e lições valiosas a serem aprendidas [Wentzel94, Frakes91]. Verifica-se que o reúso efetivo de subsistemas exige que a técnica de desenvolvimento esteja integrada a uma política de administração do processo de desenvolvimento, bem como à forma de controle dos subsistemas já existentes.

A técnica de design aqui apresentada privilegia a maximização dessas métricas. Para assegurar um alto grau de Modularidade, o sistema deve ser construído a partir de subsistemas independentes, com serviços e interfaces bem definidos. Para garantir boa Manutenibilidade, além de definir uma política de modularização do design, deve-se gerar modelos com alta comunicabilidade e uma política de documentação organizada de forma modular. A construção de um sistema portátil requer um design abstrato em relação a qualquer arquitetura de hardware e software básico. Para alcançar o objetivo de Reúso, cada sistema deve ser construído com base em subsistemas reusáveis. O grau de reúso de um subsistema [Poulin94] é função de padrões de implementação (por exemplo, generalidade suficiente para satisfazer múltiplos usuários, extraída a partir de

requisitos obtidos dos potenciais usuários) e suficiente grau de informação de suporte (documentação clara) para torná-lo facilmente empregável.

Essas características de qualidade habilitam uma gerência mais efetiva do processo de desenvolvimento. Subsistemas independentes podem ser implementados em paralelo, o que abrevia a apresentação de resultados ao cliente/usuário. Essa independência facilita o uso de uma estratégia de Desenvolvimento Incremental, que o sistema seja construído de forma gradativa, cada versão entregue constituindo um subsistema efetivamente operacional com funcionalidade agregada à versão precedente [Gomma93, Brooks87, Fairley85]. Entre as vantagens que uma estratégia desse tipo incorpora ao processo de desenvolvimento, podem ser citadas:

- adaptação gradual do usuário ao manuseio do sistema;
- minimização dos problemas relacionados à expectativa e insatisfação relacionadas com a demora do processo de desenvolvimento;
- revisão corretiva do planejamento de capacidade do hardware escolhido.

4. A Estrutura de Modelagem

Tradicionalmente, os chamados **Métodos Estruturados** têm sido orientados a atividades, buscando primordialmente a identificação das funções que constituem o sistema. Modernamente, incorporam a modelagem da informação armazenada [Maffeo92, Yourdon86, Ward84, McMenamin84].

A estrutura de modelagem preconizada [Ward85, Maffeo92] inclui três modelos, com graus de abstração diferentes, que detalham toda a concepção e implementação do sistema. Existe uma relação de precedência lógica entre esses modelos que, entretanto, não define necessariamente a ordem cronológica mais adequada para sua construção. São eles:

- **Modelo da Essência** (ME), representando o resultado da elicitação, análise e especificação dos requisitos essenciais (isto é, independentes de qualquer alternativa de implementação) do sistema em questão. Neste trabalho, não serão apresentados detalhes referentes a esse modelo, que podem ser encontrados, por exemplo, em [Sanchez94, Sanchez93, Maffeo92];
- **Modelo da Implementação** (MI), sucessor lógico do ME incorporando os requisitos essenciais ao design do sistema, caracterizado ao longo deste trabalho.
- **Modelo da Automação** (MA), sucessor lógico do MI incorporando o design de alternativa de implementação selecionada aos recursos tecnológicos de nível de abstração mais baixo (linguagem de programação, processadores, ...), pouco explorado neste trabalho.

Os produtos gerados nas diferentes etapas do processo de desenvolvimento estão ilustrados na figura 1, onde cada modelo engloba o conteúdo do modelo que o precede logicamente.

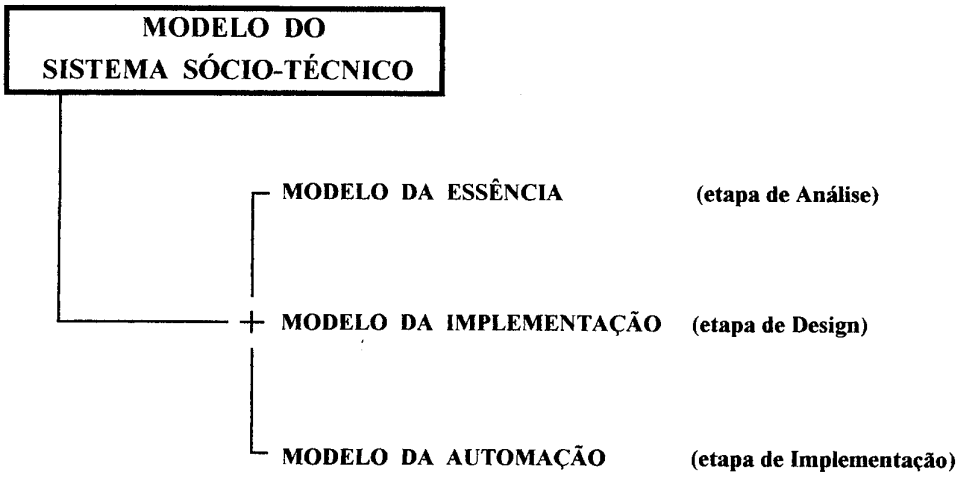


Figura 1 - Visão Estrutural do Processo de Modelagem

4.1. O Modelo da Essência

O Modelo da Essência utiliza ferramentas conceituais e técnicas de modelagem rigorosas, primordialmente gráficas, com sintaxe e semântica bem definidas, que facilitam significativamente a comunicação entre pessoas com níveis de conhecimento distintos em relação ao uso da tecnologia de computação digital. A modelagem esquemática, através de gráficos legendados refletindo visões abstratas de elementos funcionais ativos (funções), baseia-se numa notação que amplia aquela usada para construir Diagramas de Fluxo de Dados. Essa notação, cuja semântica está parcialmente formalizada em [Richter92], foi denominada ESML (Extended Systems Modeling Language) [Bruyn88, Maffeo91]. Sua eficácia fica refletida na clareza e precisão da representação obtida, como pôde ser constatado em [Clemente92, Barbosa92, Ackerman91], onde são apresentados os resultados de experimentação controlada relativa a seu emprego na modelagem de sistemas de tempo-real.

O domínio da complexidade de um problema, de modo análogo à concepção de um sistema, exige necessariamente algum esforço de segmentação/abstração. Mas, como em todo projeto de engenharia, esse esforço deve obedecer a critérios bem estabelecidos que não deixem dúvidas sobre o caminho a seguir.

A técnica aqui apresentada tem como primeiro ponto de segmentação/abstração o critério de "Essência versus Implementação" [Ward85, Maffeo92]. A modelagem da essência abstrai a forma de implementar o sistema - solução concreta de um problema computacional. Para representar adequadamente a solução abstrata, novamente utiliza-se o processo de segmentação/abstração, agora baseado em um critério de "pertinência ao sistema". Esse critério segmenta o Modelo da Essência em dois submodelos, o Modelo do Contexto e o Modelo do Comportamento, que representam, respectivamente, a definição das necessidades do ambiente externo - o enunciado do problema computacional - e uma solução possível para esse enunciado, abstraída de referências a formas de implementação.

Cada um desses dois modelos é composto de submodelos complementares, construídos a partir de enfoques diferentes sobre o ambiente externo (modelagem do contexo) e sobre o sistema (modelagem do comportamento). Esses submodelos são interdependentes e necessidades do ambiente externo, registradas no Modelo do Contexto, podem vir a ser descobertas durante a

construção do Modelo do Comportamento. Isso determina alterações no Modelo do Contexto, visando manter a consistência global do Modelo da Essência.

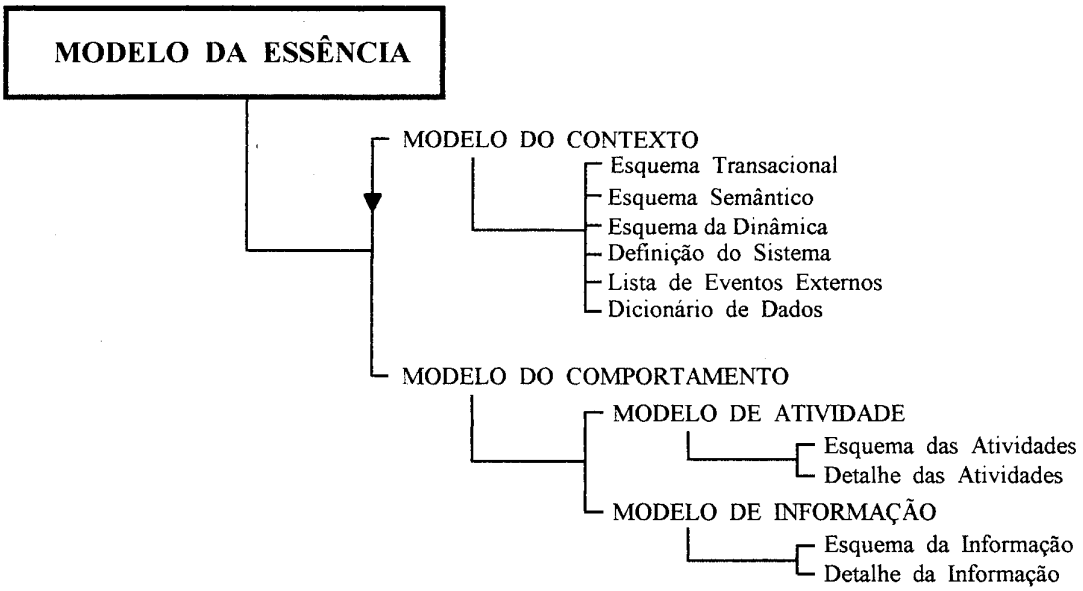


Figura 2 - Visão Estrutural da Modelagem da Essência

Assim, o **Modelo da Essência** [Maffeo92, Clemente92, Ritto91, Maffeo90] apresenta a seguinte estrutura, representada esquematicamente na figura 2 (a seta indica precedência lógica):

Modelo do Contexto, composto de:

- **Definição de Sistema**, onde são descritos os objetivos e a operação "caixa-preta" do sistema;
- **Lista de Eventos Externos**, identificando os eventos que ocorrem no ambiente externo ao sistema de controle/monitoramento - eventos externos - e que suscitam reações planejadas por parte desse sistema. Essa lista constitui um instrumento básico de modelagem das necessidades a serem atendidas pelo sistema;
- **Esquema Transacional**, que utiliza a ESML, modelando a interface do sistema com as entidades externas com as quais interage diretamente;
- **Esquema Semântico** [Maffeo90], constituído por um conjunto de hierarquias semânticas do tipo "é-um" e "é-composto", explicitando a natureza e a composição dos fluxos presentes no Esquema Transacional e relacionando-os a transações bem definidas;
- **Esquema da Dinâmica** [Clemente92], que utiliza a linguagem gráfica dos Diagramas de Estados e Transições ou Redes de Petri Compactas, modelando a dinâmica dos eventos externos de interesse para o sistema.

Modelo do Comportamento, composto de:

- **Esquema da Memória Essencial**, onde o instrumento de modelagem é o DER (Diagrama de Entidades e Relacionamentos) [Chen76], incluindo toda a parte esquemática da modelagem conceitual dos elementos funcionais passivos do sistema (tipos de entidade, seus atributos e tipos de relacionamento entre entidades);
- **Esquema das Atividades Essenciais**, que utiliza a ESML, modelando todas as atividades essenciais em um mesmo diagrama de rede, não hierárquico;

- **Representação Hierárquica das Atividades**, que utiliza a ESML, modelando uma hierarquia de atividades até o nível das atividades primitivas, estas representando as funções do sistema que reagem a ocorrências de eventos externos;
- **Diagramas de Estados e Transições**, um para cada atividade de controle primitiva, constituindo sua especificação formal;
- **Listas de Pré /Pós Condições**, uma para cada atividade operacional primitiva, constituindo sua especificação formal.

A abordagem empregada na modelagem do comportamento é "outside-in". O esquema de Atividades Essenciais é parcialmente deduzido a partir da Lista de Eventos Externos e do Esquema da Dinâmica construídos durante a modelagem do Contexto. A Representação Hierárquica das Atividades Essenciais constitui a organização hierárquica dessa modelagem funcional. O Esquema da Memória Essencial é parcialmente deduzido a partir do Esquema Semântico e do Esquema Transacional construídos durante a modelagem do contexto.

Complementando os dois modelos, constrói-se um **Dicionário de Dados** que define a informação presente na parte esquemática desses modelos.

Esses submodelos são baseados em perspectivas diferentes sobre o ambiente externo e sobre o sistema. Isso introduz uma redundância na representação global, controlada por imposição da consistência que deve garantir a precisão e não ambigüidade do modelo resultante. A verificação da consistência interna do modelo é facilitada pela construção de uma **Tabela de Verificação de Consistência** [Maffeo92] que relaciona as entradas, reações e respostas do sistema, associadas a cada evento externo.

4.2. O Modelo da Implementação

O Modelo da Implementação consiste em construir a planta que agrega características não essenciais do sistema aos requisitos essenciais definidos no Modelo da Essência. Esse modelo resulta de uma reorganização e expansão do Modelo da Essência, adicionando-se a este atividades e informações que visam superar a hipótese não realista de tecnologia ideal empregada na construção do Modelo da Essência [McMenamin84]. Dessa forma são completados os requisitos necessários ao atendimento das necessidades do usuário. O Modelo da Implementação é composto de três porções distintas:

- a definição dos **Requisitos de Concepção de Sistema**, agregando requisitos não essenciais que especificam características de desempenho, tolerância a falhas e outras. Essa definição impõe restrições à tecnologia não ideal a ser empregada na implementação e é precondicão para a escolha das arquiteturas de harware e software básico;
- a definição da **Interface Usuário-Máquina**, que determina as características interativas do sistema e sua forma de operação. Implica selecionar ou construir os equipamentos de interface com os operadores e o software a ser incorporado ao sistema para estabelecer sua interatividade;
- o **design do sistema** que o segmenta em subsistemas com um padrão de cooperação para atender aos requisitos essenciais, não essenciais e de comunicação com operadores.

A estratégia de design empregada para atingir os objetivos de qualidade especificados anteriormente recomenda a divisão do processo de design do sistema em três etapas:

- construção do **Projeto Básico**, que evidencia os **subsistemas básicos,** suas **portas** e **conexões**, bem como a eventual necessidade de concorrência e sincronismo entre os serviços alocados a cada subsistema básico;
- **especificação detalhada** de cada **subsistema básico**, que permite seu desenvolvimento autônomo e facilita sua recuperação para reúso;
- **design** de cada subsistema básico como unidade independente de desenvolvimento, envolvendo implementação e reúso;
- **configuração do sistema**, determinando-se a **configuração de software** (definição de portas e conexões) e a **configuração de hardware** (mapeamento dos subsistemas básicos e seus processos no hardware).

O **Projeto Básico**, resultado de uma ação de síntese aplicada sobre o Modelo da Essência, contém a definição dos **subsistemas básicos**, que incorpora a especificação dos serviços prestados pelo sistema associados a cada um desses subsistemas. **Serviços** são especificados a partir de agregações de Eventos Externos. **Subsistemas básicos,** definidos a partir de critérios de síntese aplicados ao Modelo do Comportamento e baseados no encapsulamento de informação e operações, englobam um conjunto resumido e bem definido de serviços passível de reúso em outros sistemas.

Os **subsistemas básicos** são **unidades de implementação** e **unidades de reúso**. Unidades de implementação que se prestam ao design, manuseio, produção e manutenção de grandes sistemas que encapsulam informação e operações de modo a garantir a inexistência de áreas de dados compartilhadas. Cada subsistema básico é uma unidade reusável de código fonte desenvolvida e mantida por uma equipe pequena e é indivisível quando reusada.

Para aumentar o poder de comunicação do Projeto Básico, o Sistema é representado como uma organização hierárquica de subsistemas com serviços alocados a cada subsistema. Cada subsistema folha da hierarquia é considerado **subsistema básico**. A segmentação do sistema em subsistemas básicos obedece à restrição de que um serviço só pode aparecer em um dado subsistema; caso algum outro subsistema necessite deste serviço deve pedi-lo ao subsistema ao qual o serviço esteja alocado. Como os subsistemas básicos encapsulam toda a informação tratada pelo sistema, nenhum dos níveis da organização hierárquica apresentará área de dados compartilhada. Um subsistema pode possuir várias portas de entrada (simples ou com resposta associada) e, portanto, prestar mais de um serviço concorrentemente.

Com o **Projeto Básico,** é possível entender o funcionamento do sistema como um todo e as opções de design que levam a um dado particionamento do sistema em subsistemas. Para sua representação é necessário empregar uma linguagem diagramática hierárquica, apoiada por uma representação de controle, também hierárquica, que represente a estrutura da decomposição do sistema e a concorrência entre subsistemas e entre serviços alocados a um dado subsistema.

O Projeto Básico especifica o conjunto de serviços pertencente a cada subsistema básico de uma forma ainda abstrata e sucinta. Entretanto, para que a implementação seja feita por equipes diferentes, cada subsistema básico necessita de uma especificação detalhada independente. Isto é, o serviço especificado no Projeto Básico ainda não estará descrito com o nível de detalhe necessário à implementação. A partir dele e do Modelo da Essência do sistema, será feita uma especificação detalhada do subsistema básico. Deve ser assegurado o cumprimento das necessidades do cliente/usuário (incorporação de todas as ações, informações e restrições especificadas no Modelo da Essência) e a comunicação via as portas determinadas no Projeto Básico. Nessa especificação, para facilitar o reúso do subsistema básico, limites e restrições devem, sempre que possível, identificar o tipo de parametrização a ser empregado na implementação [Poulin94].

A separação entre a documentação do design e a especificação de cada subsistema básico facilita a divisão do trabalho entre equipes responsáveis por um ou mais subsistemas básicos, bem como a identificação dos subsistemas básicos, já desenvolvidos, que podem ser reusados.

O design do sistema resulta em um conjunto de subsistemas básicos (SB) independentes, cuja integração será efetuada através de uma **linguagem de configuração** com a função de definir as conexões, verificar a consistência entre as portas conectadas e mapear os processos de um dado SB na Arquitetura de Hardware.

Essa visão de Engenharia de Software [Sanchez93.1] permite a divisão de trabalho no processo de desenvolvimento de grandes sistemas, além de facilitar o planejamento, a gerência do projeto de desenvolvimento e o processo de integração do sistema. Utilizar uma linguagem de configuração facilita a integração gradativa do sistema, viabilizando uma estratégia gerencial que reduz significativamente o risco de desenvolvimento de sistemas de tempo-real e de grande porte - **Estratégia de Desenvolvimento Incremental**.

A figura 3 representa esquematicamente a estrutura proposta para o Modelo da Implementação.

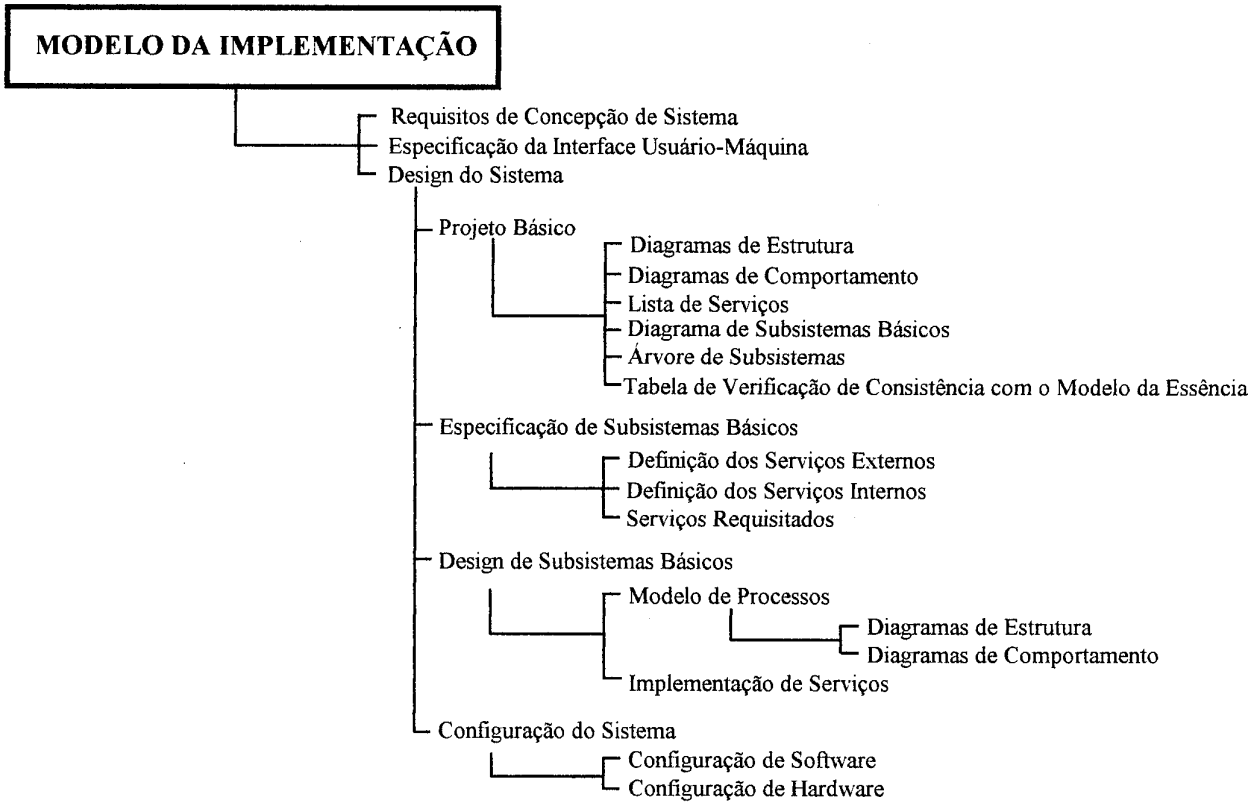


Figura 3 - Visão Estrutural da Modelagem da Implementação

Conceitos de Design - Resumo

O **sistema** é implementado por um conjunto de **subsistemas básicos**.
Um **subsistema básico** oferece e solicita **serviços** através de suas **portas de entrada/saída**.
As **portas** de um dado **subsistema básico** são ligadas a **portas** de outros **subsistemas básicos** através de **conexões**.
A **Interface** entre subsistemas básicos, constituída pelas portas e conexões, é implementada utilizando-se uma **Linguagem de Configuração de Software**.
A cada **porta de entrada de um subsistema básico** está associado um **processo**.
Os **processos** de cada **subsistema básico** são mapeados no hardware através de uma **Linguagem de Configuração de Hardware**.

4.3. O Modelo da Automação

Resulta da transformação do Modelo da Implementação, planta lógica do sistema, em planta física A porção desta planta física correspondente à parte automatizada do sistema sócio-técnico especifica o comportamento do sistema através de código fonte expresso em termos de uma linguagem de programação. Desta porção consta também a documentação correspondente ao hardware, construído durante o processo de desenvolvimento ou adquirido no mercado. Neste trabalho não serão apresentados detalhes do Modelo da Automação.

5. Modelo da Implementação - Conceitos

Nesta seção, são definidos os conceitos que constituem o fundamento da modelagem da implementação (design) e os critérios que orientam a busca da melhor estrutura para o sistema.

5.1. Objetos e Classes

O termo **objeto** será usado com sentido análogo ao de instância de um "Tipo Abstrato de Dado". A técnica de design proposta emprega os conceitos de "Encapsulamento de Dados" e de "Abstração de Dados" da Engenharia de Software, primordialmente criados para permitir modularidade acoplada a desenvolvimento autônomo de módulos ("Information Hiding") [Parnas].

O Encapsulamento de Dados envolve o conjunto Estrutura de Dados e suas Rotinas de Acesso, isto é, a estrutura de dados é apenas acessível através das operações providas pelas rotinas de acesso. Dessa forma, o "Encapsulamento de Dados", abstraída a estrutura de armazenamento, provê o conceito de objeto entendido como instância de um Tipo Abstrato de Dado [Fairley85].

Uma **classe** descreve um padrão para objetos com propriedades semelhantes (atributos), com o mesmo comportamento (operações), com os mesmos relacionamentos com objetos de outras classes e com a mesma semântica [Rumbaugh91]. É definida por:

- um **nome** que a identifica univocamente;
- uma **representação** que inclui a definição dos dados encapsulados;
- um conjunto de **operações** que definem as funções que podem ser executadas sobre os dados encapsulados.

Objetos instanciam classes e podem ter características:

- **reativas**, quando tem operações ativadas por outros objetos;
- **ativas**, quando têm existência própria e executam operações independententemente de outros objetos.

Objetos constituem uma forma alternativa para estruturar sistemas constituindo unidades autônomas de programação e de distribuição entre as quais a comunicação acontece. Pode-se dizer que objetos são "inteligentes" [Gilbert93] pois:

- conhecem seus próprios estados;
- pedem informação a outros objetos;
- informam a outros objetos sobre eventos.

Ao agruparmos os objetos em classes [Rumbaugh91], organiza-se a partir de uns poucos casos (objetos) criando-se um hospedeiro (classes) de inúmeros casos semelhantes. Atributos e operações são características de classe. Objetos reúsam o código escrito para sua classe e derivam sua individualidade das diferenças entre os valores de seus atributos e de seus relacionamentos com outros objetos.

5.2. Sistemas X Serviços

O domínio da complexidade de sistemas (devido ao porte e intrínseca, relacionada à densidade de detalhes) se dá através de técnicas de segmentação e abstração. Decorrente da recursividade do conceito de sistemas [Maffeo92], um sistema pode ser segmentado em subsistemas e cada subsistema, por sua vez, pode, eventualmente, ser considerado um sistema. Dentro do contexto aqui associado ao design de sistemas sócio-técnicos, um subsistema não é um objeto. É uma implementação de classe possuindo uma interface restrita e bem definida para que possa relacionar-se a outros subsistemas componentes do sistema sócio-técnico.

A aplicação do processo de segmentação de um sistema/subsistema em subsistemas pode ocorrer tantas vezes quanto for necessário para permitir a representação de uma mensagem com alta poder de comunicação. A modelagem de sistemas constitui a técnica de abstração que melhor trata a complexidade intrínseca de um sistema. No entanto, essa técnica de segmentação/abstração necessita de critérios que norteiem a escolha de uma ou outra solução.

Um sistema pode ser considerado como um prestador de serviços ao ambiente externo e esse enfoque determina um critério de segmentação - cada subsistema é responsável por alguns dos serviços prestados pelo sistema. Um **Serviço** é um grupo de funções (análogo a um conjunto de operações de uma classe) inter-relacionadas que compartilham algum propósito comum; em particular, gerenciam uma estrutura de dados originária da estrutura encapsulada pela classe da qual deriva o subsistema. O Sistema é modelado como um conjunto de subsistemas evidenciando-se, na sua segmentação, a estrutura de dados alocada a cada subsistema e o conjunto de serviços a ele atribuído. A dinâmica do sistema é caracterizada pela ativação desses serviços.

Um serviço pode ser:

- **externo**: quando executado a partir de uma requisição realizada por outro subsistema ou por entidade externa ao sistema;
- **interno**: quando iniciado a partir de condições limites ou do decorrer de um período de tempo, executado independentemente de requisições.

Subsistemas oferecem e requisitam serviços de outros subsistemas. Muitos desses serviços podem ser concorrentes. A cada **serviço concorrente** está associado um **Processo**, cuja ativação pode ser periódica ou em atentimento a uma requisição de serviço.

5.3. Interfaces

O sistema sócio-técnico comunica-se com entidades externas através de uma **interface** por onde fluem controle e dados. Por exemplo, quando um operador faz incidir um fluxo de dados sobre o sistema, dois tipos de informação ocorrem simultaneamente:

- a ordem de executar um comando (conteúdo de controle do fluxo de dados);
- os dados de que o sistema necessita para operar e que não existem internamente ao sistema.

Utilizando a recursividade do conceito de sistema, pode-se dizer que seus subsistemas interligam-se também através de interfaces. **Interfaces** portanto podem ser definidas como o conjunto de dados e controle através do qual um subsistema/sistema comunica-se com o ambiente externo a ele (outros subsistemas/entidades externas).

5.4. Portas

Visando proporcionar nível elevado de reúso, subsistemas devem ser concebidos de modo que cada um tenha conhecimento apenas de si próprio, isto é, sem incorporar referências a outros subsistemas. Para obter essa independência, cada subsistema deve possuir **portas,** de entrada e saída de dados e controle, que definam o protocolo de comunicação com o ambiente externo ao subsistema, regulando o **envio/recebimento de mensagens**. Essas mensagens são transportadas por **interfaces** composta de **canais de comunicação** que conectam **portas** de subsistemas. A comunicação e o sincronismo entre subsistemas se dá através dessa interface.

A definição precisa de um subsistema exige a especificação de:

- a(s) entrada(s);
- a(s) saída(s);
- como, a partir da(s) entrada(s), pode-se produzir a(s) saída(s); incluindo eventuais restrições que o sistema deve satisfazer, isso significa a especificação de uma relação, no sentido matemático do termo, que pode ser implementada alternativamente por mais de um procedimento.

No método de design aqui proposto, a definição das portas de um subsistema satisfaz o requisito de especificar entradas e saídas. A cada porta associa-se um tipo de variável que a descreve, isto é, uma estrutura descrita a partir de dados atômicos. Toda definição, design e automação do subsistema deve fazer referência apenas a suas portas e nunca a outros subsistemas com os quais se comunique. Através dessas portas são efetivamente ativados os serviços que o subsistema presta ao mundo exterior, enviados os dados necessários à execução de um serviço requisitado e recebidas as respostas correspondentes a sua execução. Desta forma, torna-se possível satisfazer o requisito de reúso pois cada subsistema só possui conhecimento de suas próprias portas e, desde que seja respeitado o protocolo especificado para elas, subsistemas podem ser empregados em outros sistemas sem sofrer alterações, comunicando-se com subsistemas diferentes daqueles empregados na configuração do sistema para o qual foi desenvolvido. Essas características de comunicação entre subsistemas possibilitam também configurar, em um mesmo sistema, diversos subsistemas implementados por linguagens de programação diferentes.

O **protocolo** associado a cada porta é definido por:

- sentido principal do fluxo de mensagem através da porta (requisição/prestação de serviço);
- característica do tipo de serviço requisitado/prestado através dessa porta (simples ou com resposta associada);
- conteúdo das mensagens (tipos de dados enviados/recebidos) que chegam/saem da porta.

O **sentido principal do fluxo de mensagem** caracteriza uma porta em:

- **de entrada**: que canaliza o estímulo recebido pelo subsistema para o serviço externo pertinente desse subsistema;
- **de saída**: de onde parte uma requisição de serviço externo feita a outro subsistema.

Uma **requisição de serviço** é reconhecida pelo subsistema através do recebimento de uma mensagem em uma porta de entrada. Qualquer requisição de serviço pode ter uma resposta associada; isto é, se a uma porta de entrada chegar uma requisição de serviço, através dela pode sair uma informação associada à resposta produzida pelo serviço.

A requisição de um serviço é efetuada através de uma porta de saída do subsistema cliente conectada através de um canal de comunicação a uma porta de entrada do subsistema servidor. Essa forma de interação entre subsistemas estabelece a dinâmica global do sistema integrado. Considere uma porta **a** de saída do subsistema **A** conectada a porta **b** de entrada do subsistema **B;** o subsistema cliente **A** requisita um serviço ao subsistema servidor **B**. A requisição do serviço é implementada pelo envio de mensagem através da porta **a** do subsistema cliente e seu recebimento através da porta **b** pelo subsistema servidor; isto é, pela implementação do canal de comunicação entre essas portas.

A **característica do tipo de serviço** determina o sincronismo entre subsistemas e classifica as portas em:

- **porta de entrada** à qual associa-se a chegada de dados e a ativação de um serviço do subsistema;
- **porta de saída** à qual associa-se o envio de dados juntamente com a requisição de serviço a um subsistema servidor com uma porta de entrada ligada à porta de saída do subsistema cliente;
- **porta de entrada com resposta associada** à qual associa-se a chegada de dados e a ativação de um serviço do subsistema servidor e, após o término de execução do serviço, o envio de resposta ao subsistema cliente. Esse envio é feito pela mesma porta em que foi recebida a requisição de serviço.
- **porta de saída com resposta associada** à qual associa-se o envio de dados e a requisição de serviço a um subsistema servidor que os recebe através de uma porta de entrada com resposta associada. O subsistema cliente que requisita serviço por uma porta desse tipo pode continuar executando até necessitar da resposta. A recepção da resposta é feita pela mesma porta em que foi enviada a requisição de serviço.

A **estrutura das mensagens** determina um padrão de compatibilidade para portas conectadas: é única a estrutura de informação que transita através de um canal de comunicação para cada sentido de fluxo. Deve respeitar as seguintes regras:

- a estrutura da mensagem enviada de uma porta de saída a uma porta de entrada a ela conectada deve ser única;
- a estrutura da mensagem recebida em uma porta de entrada com resposta associada deve ser a mesma que a enviada pela porta de saída com resposta associada a ela conectada;

- a estrutura da mensagem enviada como resposta de uma porta de entrada com resposta associada deve ser a mesma que a recebida pela porta de saída com resposta associada a ela conectada.

5.5. Interação entre Subsistemas

A interação entre subsistemas, para todos os níveis de decomposição, organiza-se através de uma **interface** constituída de **portas** e **canais de comunicação**. Podem ser utilizadas portas agregadas, denotando um conjuto de tipos diferentes de portas, recurso empregado para a representação hierárquica em termos de subsistemas agregados: uma porta agregada é uma abstração que simboliza o conjunto de portas reais associadas aos subsistemas componentes do subsistema agregado.

Mais de uma porta de entrada em um mesmo subsistema revela a possibilidade de requisição simultânea dos serviços a elas associados caracterizando uma concorrência na execução desses serviços; isto é, serviços independentes prestados por um subsistema são concorrentes. Um subsistema é implementado por um ou mais **processos** onde a toda porta de entrada (simples ou com resposta associada) estará associada uma **instância de processo** ou mais de um processo no caso de subsistemas agregados. Várias portas de saída (simples ou com resposta associada) podem estar associadas a um único processo pois, para executar o serviço alocado a essse processo, o subsistema pode requisitar serviços prestados por outros subsistemas. A configuração de um sistema pode contar com mais de uma instância de um mesmo subsistema.

Em princípio, serviços ativados através de portas de entrada simples não causam espera nos subsistemas que os requisitam. Ficará a cargo da interface depositar a mensagem na porta de entrada do subsistema responsável pela execução do Serviço, dispondo esta porta de um certo enfileiramento de mensagens. Visando solucionar casos extremos (fila de mensagens cheia), a implementação da interface deve permitir duas opções:

- o subsistema requisitante ficará em estado de espera no caso de não existir espaço de armazenamento para a mensagem;
- a mensagem enviada será perdida.

Esses casos devem ficar registrados em tempo de execução de modo a permitir que o desenvolvedor/mantenedor do sistema identifique que os tempos de execução envolvidos na relação produtor/consumidor existente entre os subsistemas não está sendo satisfatória.

Serviços ativados através de uma porta de entrada com resposta associada devem, ao final de sua execução, produzir a mensagem de saída correspondente à resposta. Este tipo de conexão assemelha-se ao **Remote Procedure Call** das Linguagens de Programação. No entanto, o Processo cliente que requisita o serviço só fica suspenso ao requisitar explicitamente a resposta. Ocorre da seguinte forma, ilustrada na figura 4:

- uma porta P1 de entrada com resposta associada de um subsistema A está conectada a uma porta P2 de saída com resposta associada de um subsistema B;
- o subsistema B pede um Serviço ao subsistema A, colocando uma mensagem na porta P2. Essa mensagem será entregue na porta P1 (pela interface). O Processo de B que trata a porta P2 pode continuar executando até necessitar da resposta;
- ao necessitar da resposta, o Processo de B requisita uma mensagem na porta P2. Caso essa mensagem, correspondente à resposta à requisição de serviço feito anteriormente, ainda não esteja disponível, esse Processo é suspenso, e assim permanece à espera de resposta;
- o subsistema A executa o Serviço correspondente à mensagem recebida na porta P1 e coloca a mensagem que contém a resposta (saída) na porta P1. Essa mensagem será entregue na porta

P2 de B. Nesse instante, o Processo que trata a porta P2, caso esteja suspenso, volta a estar pronto para executar;

- o processo do subsistema A, que trata a porta P1, suspenso após ter produzido a mensagem de resposta, espera a chegada de nova mensagem. Enquanto isso não ocorrer, esse Processo permanece suspenso.

Portas desse tipo, onde existe a suspensão do processo a ela associado, devem, por questões de segurança, prover ao processo uma mensagem de erro caso o serviço requisitado exceda a um tempo máximo para produzir a resposta. Isso permite que a aplicação trate problemas relacionados a falhas e faltas ocorridos em tempo de execução.

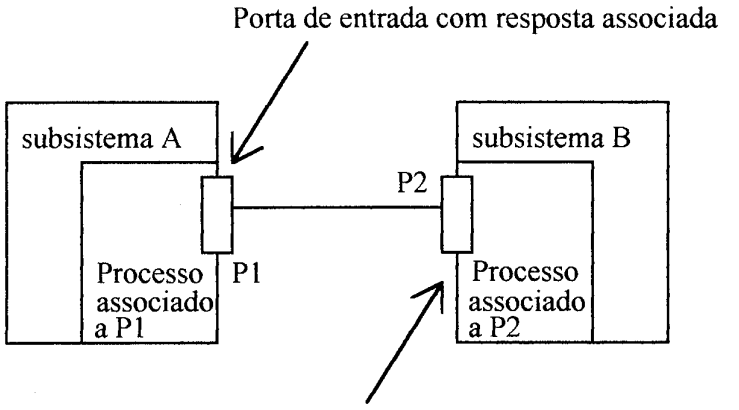


Figura 4 - Comunicação e Sincronismo entre Subsistemas

5.6. Configuração do Sistema

Subsistemas prontos e testados precisam ser integrados e, para isso, as **interfaces** devem ser implementadas e os **processos** que compõem cada **subsistema** precisam ser mapeados na Arquitetura de Hardware. O modelo dessa integração define a **configuração do sistema**. Para assegurar portatilidade, a configuração do sistema deve ser feita através de uma **linguagem de configuração** e manter-se independente da implementação dos subsistemas. A configuração do sistema é um módulo de software autônomo e alterações de Arquitetura de Hardware e Software implicam modificações introduzidas nesse módulo.

Trabalhos recentes na área de desenvolvimento de sistemas distribuídos confirmam os benefícios de manter o modelo da configuração do sistema independente dos modelos dos componentes de software que implementam as funções do sistema [Magee94, Kramer92, ESPRIT89]. Uma especificação de configuração independente pode ser usada tanto para a descrição das arquiteturas de hardware e de software básico [Kramer85] quanto para a descrição da arquitetura de software da aplicação. Isso aponta um caminho de segmentação vertical em duas camadas do software que está sendo desenvolvido, conforme ilustra a figura 5. A camada de nível mais baixo (**interface de compatibilização**) depende somente das camadas inferiores (software básico e hardware) e oferece uma interface padrão de serviços prestados para a camada de nível mais alto (**aplicação**). O software desta última (que implementa os requisitos funcionais da

aplicação) pode, então, ser projetado com a portatilidade desejada aumentando também o nível de abstração do design.

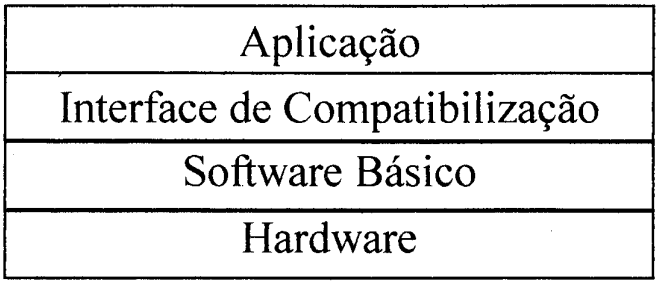


Figura 5 - Segmentação Vertical do Software

A **interface de compatibilização** oferece para a camada de aplicação um conjunto de serviços que permitem:

- definir conexões entre portas;
- estabelecer a configuração física dos processos de cada subsistema componente do sistema;
- estabelecer a comunicação entre subsistemas pelo envio/recebimento de mensagens através da interface definida.

O conjunto de serviços oferecidos pela interface de compatibilização não dependem da aplicação, dependem isto sim das características de software básico, hardware e dos conceitos acima expostos empregados no design. Portanto, essa interface pode ser reusada para várias aplicações que façam uso das mesmas camadas inferiores.

A segmentação vertical impõe um ônus relacionado ao desempenho de comunicação/sincronismo entre subsistemas pois ela privilegia a simplificação do desenvolvimento do software de aplicação e sua independência em relação à arquitetura de hardware. Essa simplificação é justificada pois o custo do processo de desenvolvimento de software tem sido o fator dominante no custo do desenvolvimento de sistemas. Na década de 80 estava em 45% do total do projeto, em 1989 situava-se em torno de 80% desse total [ESPRIT89].

Resumindo, o design do software de aplicação, abstraído da arquitetura de hardware e software básico, incorpora duas camadas de software: uma interface de compatibilização e uma camada de aplicação. A definição da interface de compatibilização, que é apenas decorrente dos requisitos de comunicação/sincronismo de processos e das características do sofware básico e do hardware, permite a construção de um software de aplicação efetivamente portátil e reconfigurável (não necessariamente em tempo de execução). Isso, na opinião dos autores, facilitaria a solução de problemas de desempenho bem como a tarefa de manutenção para a modernização do hardware.

6. O Modelo da Implementação

Neste item serão estabelecidas técnicas para a definição da arquitetura do software na camada de aplicação - O Modelo da Implementação do Sistema. Essas técnicas devem mapear rigorosamente a especificação de requisitos (ações e informações) contidas no Modelo da Essência para o design do sistema, garantindo completeza e integridade.

6.1. Requisitos de Concepção de Sistema

Primeiramente, o Modelo da Implementação deve complementar a definição dos Requisitos Essenciais, invariantes em relação a alternativas de implementação, através da agregação de requisitos não essenciais - Requisitos de Concepção de Sistema (RCS). Esses requisitos especificam exigências relacionadas ao sistema e ao Ambiente Externo que não façam parte do Modelo da Essência ou caracterizem a Interface Usuário-Máquina. Basicamente, os RCS devem incluir especificações relativas a Desempenho, Tolerância a Falhas, Ambiente de Operação (Temperatura, Umidade e Vibração), Segurança, Empacotamento (Dimensões do Equipamento) e Consumo de Energia.

Esses requisitos, não funcionais, restrigem o universo de alternativas de implementação e incorporam restrições impostas pela tecnologia disponível. Constituem um conjunto especificações importantes para o usuário que requerem registro formal antes de ser iniciada efetivamente a implementação. Assim, constituem base para a especificação das arquiteturas de hardware, software básico e da aplicação que garantam a qualidade para aceitação do produto final. Regras para construção do RCS encontram-se melhor especificadas em [IEEE84].

Os RCS podem ser classificados em tipos referentes a:

- condições de operação;
- tecnologia a empregar.

Os requisitos de condições de operação devem especificar as necessidades do usuário para a operação do sistema sob as seguintes condições:

- **falhas e faltas**: o tipo de tolerância que o sistema deve apresentar na ocorrência de falhas (erros do sistema tratados, por exemplo, através de computação redundante em mais de um processador e votação do resultado) e faltas (defeito em hardware ou sensor/atuador tratado, por exemplo, através de funcionamento degradado)
- **limites operacionais** de execução referentes, por exemplo, a volume de informação armazenada/processada, desempenho, segurança de acesso às informações, disponibilidade e confiabilidade - MTBF (tempo médio entre falhas), MTTR (tempo médio de reparo); características ambientais (temperatura, umidade, vibrações, interferência eletro-magnéticas, consumo de energia, limitações de peso/tamanho, ruído, radiação, poluição,...); características de manutenção em primeiro escalão (substituição rápida de placas com equipamento energizado).

Os requisitos referentes à tecnologia a empregar são normalmente decisões da equipe de desenvolvimento em função de outros requisitos não essenciais tais como desempenho e tolerância a falhas, entre outros. O cliente/usuário pode influenciar essas decisões já que impactam a utilização/manutenção do sistema (treinamento do usuário no uso de sistemas e/ou em uma dada tecnologia de manutenção) e/ou o custo do sistema (compra de novos compiladores, emuladores etc.). Esses requisitos devem abranger:

- hardware (por exemplo: o Sistema deve ser desenvolvido em Arquitetura Distribuída.)
- software básico (por exemplo: o Sistema Operacional deve ser X pois a equipe de desenvolvimento já o conhece e a Linguagem de Programação deve ser Y pois a empresa já possui o compilador)
- fatores relacionados a custo/cronograma de desenvolvimento, custo do sistema e sua manutenção futura podem exigir a participação do cliente/usuário no processo decisório (por exemplo: restrições quanto a fornecedores de hardware/software básico; restrições quanto à compra de novos compiladores e ambientes de desenvolvimento; restrições que determinem o

uso de pacote disponível no mercado complementado pelo desenvolvimento de partes especificadas no Modelo da Essência e não tratadas pelo pacote).

6.2. Interface Usuário-Máquina

O Modelo da Implementação inclui a definição da Interface Usuário-Máquina (IUM), a qual certamente possui várias alternativas de implementação. A escolha de uma ou outra alternativa é função principalmente das necessidades do usuário em relação à operação do sistema.

A especificação da IUM deve satisfazer os seguintes propósitos:

- estabelecer uma filosofia de entrada de comandos e apresentação de informações - respostas do sistema;
- determinar equipamentos a serem empregados na interface, tais como tipos de teclados, terminais, consoles especiais etc.;
- definir os comandos aos quais o sistema deve responder e relacioná-los a sua forma de entrada no sistema (por exemplo: menus ou teclas dedicadas);
- para o conjunto de comandos selecionados através de menus, especificar a árvore de menus que permita sua seleção;
- para comandos que possuam parâmetros, determinar tipos de telas de entrada de parâmetros (caixas de diálogo) ou qualquer outra forma de seleção que garanta o uso apropriado do sistema pelo operador. Para cada parâmetro, devem ser estabelecidos valores predefinidos e a possibilidade de escolha da unidade característica de uso.

Trabalhos na área de Abstract Data Views (ADV) [Cowan95] comprovam a possibilidade e evidenciam as vantagens de especificar independentemente o design da IUM em relação ao design da aplicação. Os conceitos de ADV foram aplicados à implementação da IUM e, como pode ser observado em [KP88, HIL92, CCCL93], o design da interface usuário-máquina é singularizado de modo a facilitar o reúso de sua implementação. Esses conceitos permitem também a associação de diversas interfaces a uma mesma aplicação [CILS93a, CILS93b], provendo reúso independente da interface e da aplicação. Os trabalhos nessa área afirmam [Cowan95] que os objetos que implementam a aplicação (ou Tipos Abstratos de Dados) não necessitam conhecer a forma pelo qual seu estado é apresentado ao usuário ou como o usuário interage com essa representação. Conhecimento do mundo exterior pode portanto ser provido ao sistema de forma independente daquela que caracteriza a interação entre a interface e os objetos da aplicação. O método de implementação aqui proposto, empregando ferramentas diferentes de modelagem também prescreve a separação entre IUM e aplicação. O design do sistema deve derivar um conjunto de subsistemas que trata apenas da aplicação e outro conjunto (disjunto) de subsistemas que tratam apenas da interface com o cliente/usuário.

6.3. Projeto Básico

Subsistemas são vistos como unidades prestadoras/solicitantes de serviços que, integrados, compõem o sistema. O **Projeto Básico** é um modelo que reflete a organização hierárquica dos subsistemas que compõem o sistema e deve ser capaz de representar:

- **subsistemas componentes** e os **serviços** a eles alocados;
- **portas** de cada subsistema e seus tipos;
- **conexões** entre portas;
- **dinâmica** de interação entre serviços;

representando a interação entre subsistemas através de suas portas, as conexões dessas portas via canais de comunicação e, cooperativamente, esses subsistemas realizam os serviços previstos para todo o sistema.

6.3.1. Derivando Serviços a Partir do Modelo da Essência

O Modelo da Essência contém a Lista de Eventos Externos que modela as necessidades a serem atendidas pelo sistema. A Descrição de Operação identifica quais desses Eventos Externos em termos de **assuntos** identificados no Ambiente Externo que merecerão tratamento pelas funções do sistema. Ao classificar em Lista de Eventos Externos e a Descrição de Operação, esses assuntos permitem identificar agregações dos serviços prestados pelo sistema ao ambiente externo.

A partir do Modelo do Comportamento podem ser derivados, por um processo de síntese, os subsistemas básicos e as Atividades Essenciais a eles alocadas. A relação **Serviços x Eventos Externos** e **Subsistemas Básicos x Atividades Essenciais** determina quais serviços estão alocados a que subsistema básico já que o tratamento de Eventos Externos corresponde à execução de Atividades Essenciais.

6.3.2. Derivando Subsistemas Básicos a Partir do Modelo da Essência

Baseando-se no Esquema da Memória Essencial, no Esquema de Atividades Essenciais e na Organização Hierárquica de Atividades representada no Modelo do Comportamento, é possível propor uma configuração de **subsistemas básicos** para o sistema. Cada subsistema básico corresponde a uma alternativa de implementação para uma classe de objetos manipulada pelo sistema, isto é, uma estrutura de informação encapsulada com um conjunto de operações atuando sobre essa estrutura.

6.3.2.1. Classes x Subsistemas Básicos

O caminho que conduz, do Modelo da Essência, ao Modelo de Subsistemas Básicos passa pela determinação das classes dos objetos que compõem o sistema. Levando em conta considerações de desempenho e características do ambiente externo, o método proposto neste trabalho adota, para classes, uma alternativa de implementação - subsistemas básicos - diferente da alternativa orientada para objetos (vide figura 6). Essa escolha, entretanto, não prejudica o reúso desses subsistemas.

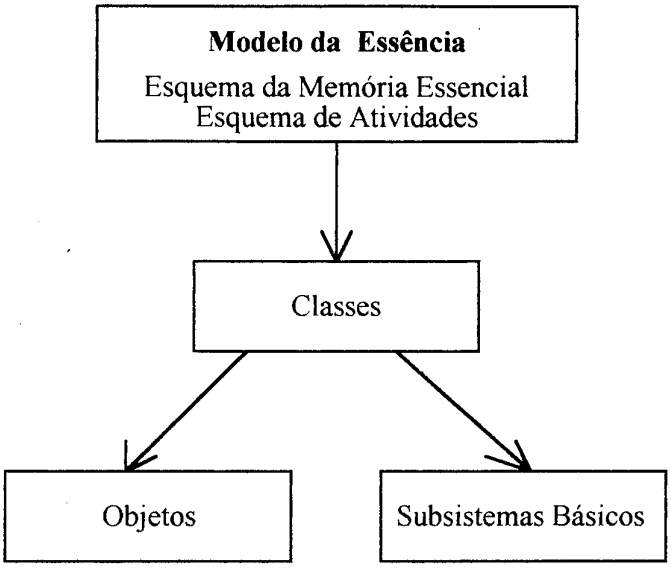


Figura 6 - Alternativas para síntese do Modelo da Essência: objetos ou subsistemas básicos

Considere um sistema em que uma de suas funções é acompanhar alvos detectados através de um radar. A classe de *Acompanhamento de Alvo por Radar* será componente do sistema.

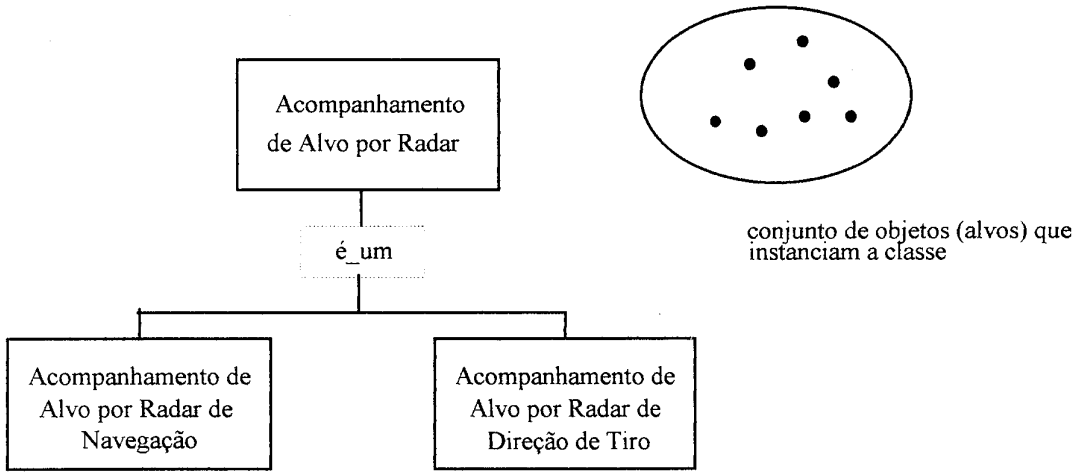


Figura 7 - Hierarquia de Classes

Conforme pode ser visto na figura 7, essa classe é uma generalização de classes específicas como as que realizam essa função através de um Radar de Navegação (com a função de acompanhar vários alvos simultaneamente) ou através de um Radar de Direção de Tiro (com a função de acompanhar um alvo de cada vez).

Para a subclasse especializada *Acompanhamento de Alvo por Radar de Navegação*, a implementação por objetos daria origem a um objeto por alvo acompanhado - o objeto correspondendo a uma instância da classe - e, portanto, a pelo menos um processo concorrente em execução para cada alvo.

Ao implementar essa classe através de um subsistema básico, são consideradas as próprias características seqüenciais da extração de informação de cada alvo pelo Radar. Ao invés de serem criados processos em número igual ao dos alvos que estão sendo acompanhados, são criadas apenas instâncias distintas da estrutura de dados que armazena os valores dos atributos da classe. O subsistema básico gerencia esse conjunto de estruturas de dados (por exemplo, um vetor de estruturas de dados onde cada alvo acompanhado esteja associado a um índice) e o acompanhamento dos alvos é realizado por um único processo que, ao tratar um alvo, atualiza a instância de estrutura a ele correspondente. Essa implementação implica o tratamento seqüencial da informação de alvos.

O conceito de porta facilita essa implementação. Associada ao subsistema básico de acompanhamento de alvos existe uma porta de entrada (ou entrada com resposta associada) onde são depositadas as informações de alvos extraídas do radar. O enfileiramento dessas informações, para tratamento seqüencial, é realizado pela própria porta.

Essa técnica de implementacão em nada altera a gereneralidade do conceito de classes. O número de objetos gerenciados por um mesmo subsistema básico pode ser determinado em tempo de geração do sistema ou, dinamicamente, em tempo de execução.

Em nenhum momento essa técnica inibe o uso de mais de uma instância do subsistema básico na configuração do sistema. O uso de mais de uma instância será necessário sempre que:

- o sistema tratar mais de um sensor/atuador do mesmo tipo;
- para garantir o atendimento aos Requisitos de Concepção do Sistema, for desejável dividir a carga de processamento por mais de um processador.

6.3.2.2. Identificando Subsistemas Básicos

A identificação das classes relevantes tem como ponto de partida o Esquema da Memória Essencial, onde estão evidenciados todos os Tipos de Entidade armazenados pelo sistema. Cada Tipo de Entidade (ou porção entidade de um Tipo de Relacionamentos com Atributos) corresponde a um depósito interno. A primeira etapa para a obtenção das classes associa uma estrutura de informação encapsulada em classe a cada Tipo de Entidade ou Porção Entidade de um Tipo de Relacionamento com Atributos presente no Esquema da Memória Essencial - ou, equivalentemente, a cada depósito presente no Esquema das Atividades Essenciais.

Em seguida, são definidas as operações dedicadas a cada uma dessas estruturas, o que pode ser feito inicialmente a partir do Esquema de Atividades Essenciais e refinado posteriormente considerando-se as Listas de Pré- e Pós-condição. Cada estrutura de informação corresponde a um depósito interno e as operações associadas a cada classe são extraídas do conjunto de Atividades Essenciais que fazem acesso à estrutura (depósito interno) por ela encapsulada. Todas essas atividades são candidatas ao encapsulamento pela classe.

As Atividades Essenciais podem ser classificadas com relação ao tipo de acesso que fazem a um dado depósito interno:

- Atividade Primária: faz acesso de escrita ao depósito para inserção ou remoção de entidades;
- Atividade Secundária: faz acesso de escrita ao depósito apenas para alteração do estado de entidades;
- Atividade Fundamental: faz apenas acesso de leitura ao depósito.

Normalmente uma Atividade Essencial faz acesso a mais de um depósito interno e a decisão relativa a qual classe deve encapsular essa atividade baseia-se:

1. em uma tabela onde cada linha corresponde a uma estrutura de informação e cada coluna reflete um tipo de acesso (escrita para inserção ou remoção de entidades, escrita e leitura para alteração de estado de entidades, só leitura) ao depósito correspondente a essa estrutura. A entrada da tabela enumera as Atividades Essenciais que efetuam o acesso definido pela coluna ao depósito correspondente à estrutura de informação definida pela linha;
2. na alocação inicial das atividades que atuam sobre cada estrutura de informação à classe correspondente, segundo os seguintes critérios:
   - uma atividade primária relativa a uma dada estrutura de informação deve ser instalada na classe que encapsula essa estrutura;
   - uma atividade secundária relativa a uma dada estrutura de informação, que não seja atividade primária relativa a qualquer outra estrutura, deve ser instalada na classe que encapsula aquela estrutura;
   - uma atividade fundamental relativa a uma dada estrutura de informação, que não seja atividade primária ou secundária relativa a qualquer outra estrutura, deve ser instalada na classe que encapsula aquela estrutura;
3. numa revisão da alocação inicial pois essa alocação geralmente apresenta:
   - uma mesma atividade essencial instalada em mais de uma classe, pelo fato de fazer acessos de mesmo tipo a mais de um depósito interno;
   - atividades essenciais instaladas em classes onde o objetivo da atividade não é relacionado ao objetivo do armazenamento da informação: por exemplo, uma atividade primária usa a inserção/remoção de informação apenas como forma de pedir um serviço a uma atividade secundária ou fundamental, esta sim relacionada ao objetivo do armazenamento da informação;

     Essa revisão deve estabelecer a coesão funcional como critério básico para instalação de Atividades Essenciais em classes: uma Atividade Essencial cujo objetivo esteja relacionado com o objetivo do armazenamento de uma dada estrutura de informação deve ser instalada na classe que encapsula essa estrutura. Assim sendo:
     - deve ser removida da classe onde foi inicialmente instalada toda atividade primária cujo objetivo não esteja relacionado ao objetivo de armazenamento da estrutura de informação que a classe encapsula. Caso a atividade removida não seja primária em relação a qualquer outra estrutura de informação, deve ser instalada na classe que encapsula uma estrutura em relação à qual seja secundária, ou na ausência desta, fundamental;
     - devem ser buscadas agregações que permitam a uma classe encapsular estruturas de informação associadas a tipos de entidades associados por tipos de relacionamentos no Esquema da Memória Essencial juntamente com (parte de) o conjunto de atividades essenciais que fazem acesso aos depósitos internos correspondentes;
4. num refinamento da revisão proposta no ítem 3, caso ainda subsistam atividades essenciais instaladas em mais de uma classe. Nesse caso, deve ser identificado o objetivo principal da atividade, que deve ser instalada na classe que encapsula a estrutura de informação mais compatível com esse objetivo. O critério de "principal objetivo" da atividade essencial a ser instalada mantém e reforça o critério básico de alocação a classes, a coesão funcional;
5. na alocação da cada atividade de controle à classe que encapsule o maior subconjunto de atividades operacionais por ela controladas. Também aqui, o critério busca maximizar a coesão funcional.

Um exemplo genérico desse tipo de síntese aplicado a um Modelo do Comportamento abstrato ilustra esse procedimento. Considere um Esquema de Atividades Essenciais e o Esquema da

Memória Essencial associado, ambos ilustrados nas figuras 8 e 9 respectivamente. Na figura 9, A, B e C representam tipos de entidade e R1 e R2 representam tipos de relacionamento do diagrama de entidades-relacionamentos que modela a Memória Essencial do sistema. Cada entidade dos tipos A, B e C corresponde a um registro dos depósitos contidos na figura 8, respectivamente DI1, DI2 e DI3. A única imposição sobre o Esquema de Atividades Essenciais da figura 8 é estrutural: além da correspondência entre depósitos e tipos de entidade, as atividades e os acessos a depósitos devem permitir a instanciação dos elementos funcionais passivos contidos na figura 9.

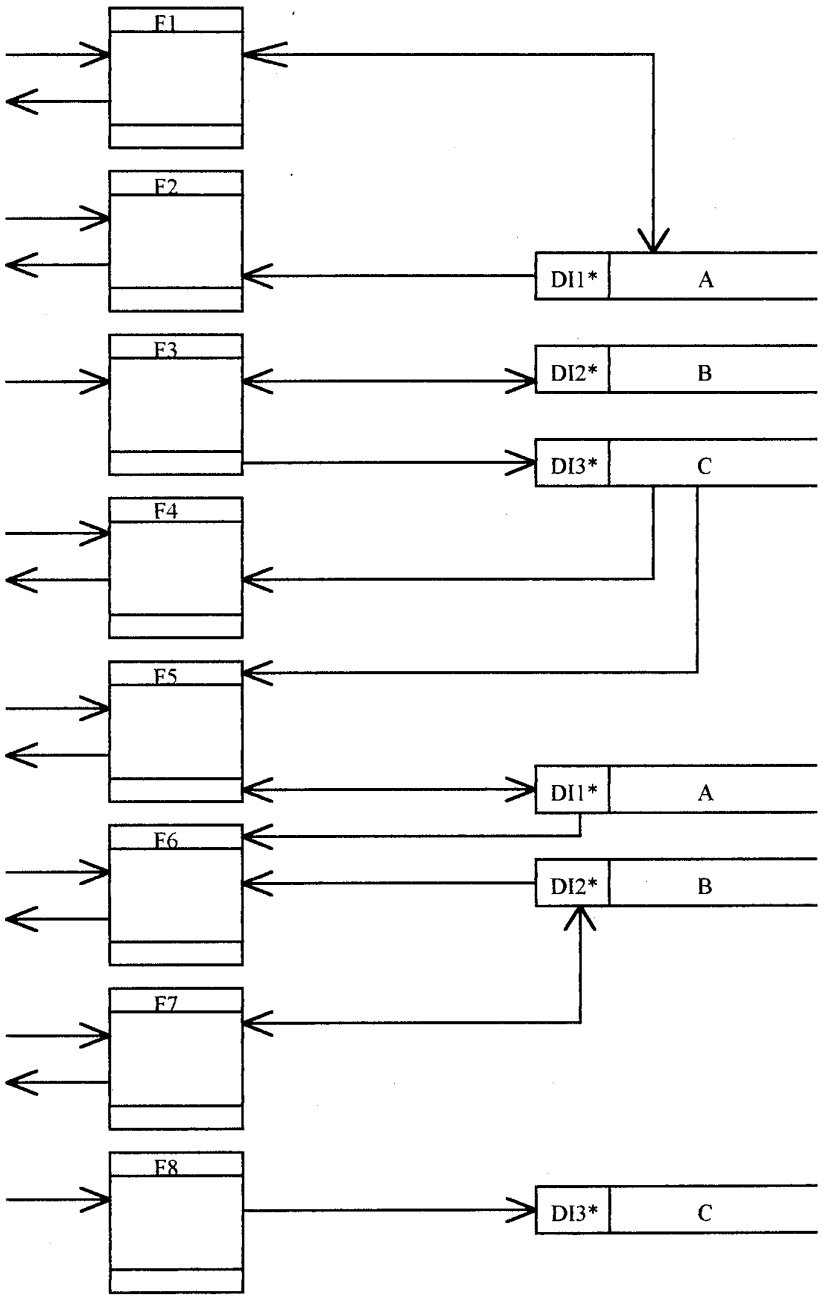


Figura 8 - Esquema de Atividades Essenciais

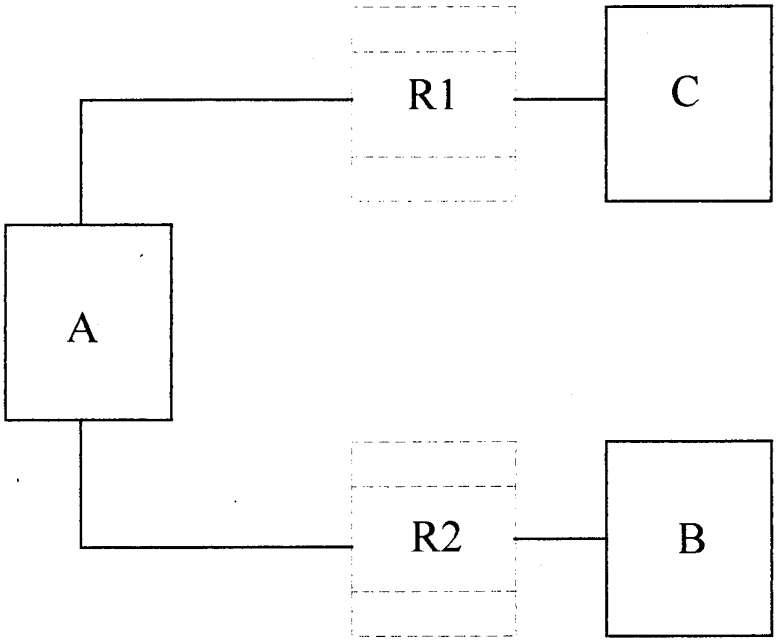


Figura 9 - Esquema da Memória Essencial

A tabela que determina a prioridade de alocação das Atividades Essenciais em classes encontra-se representada na figura 10. A semântica associada à notação empregada na figura 8 não permite distinguir atividades primárias e secundárias e, em conseqüência, essa tabela não apresenta a estrutura correspondente. O conjunto resultante de subsistemas básicos é apresentado na figura 11. Pode-se notar que:

- de um ponto de vista puramente estrutural, a alocação de F6 a S2 é arbitrária; se considerada apenas a coesão pelo tipo de acesso, essa atividade poderia ser alocada a S1.
- o número de portas da interface de cada subsistema com o ambiente externo é determinado pela concorrência existente entre os serviços prestados às entidades externas.

Critério Classe	acesso escrita inserção/remoção	acesso leitura
classe A	F1;F5	F2;F6
classe B	F3	F6
classe C	F8	F4;F5

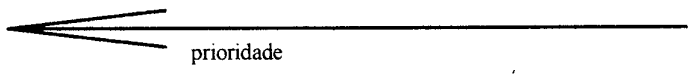


Figura 10 - Tabela de Encapsulamento de Atividades Essenciais

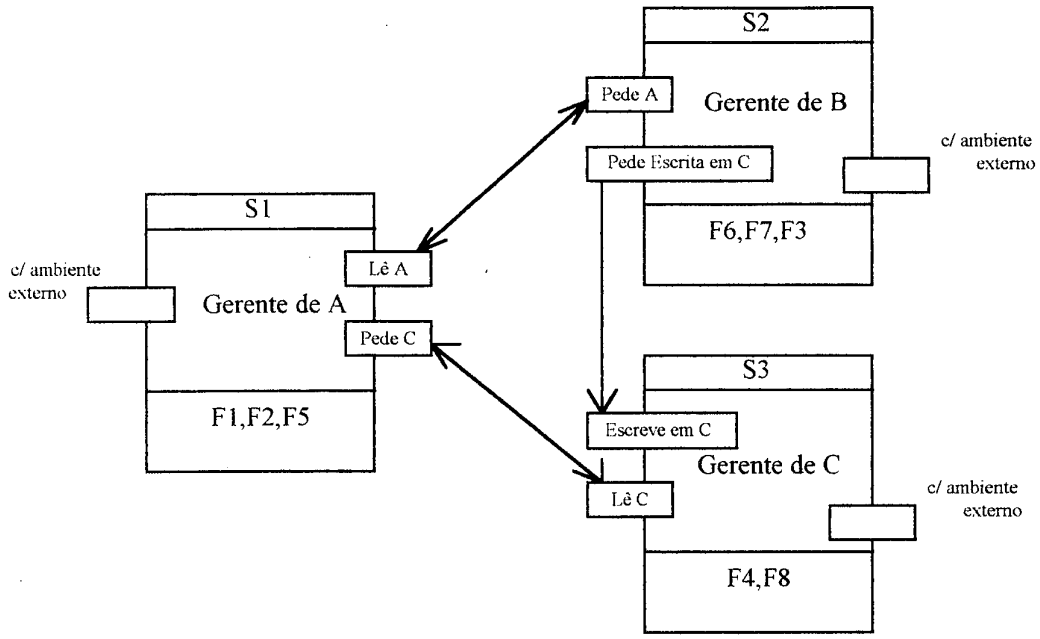


Figura 11 - Identificando Subsistemas Básicos

O mapeamento foi realizado segundo os seguintes passos:

1) associa-se um subsistema básico a cada Tipo de Entidade. Esse subsistema deve corresponder à classe que encapsula a estrutura de informação definida pelo Tipo de Entidade;
2) a alocação das Atividades Essenciais às classes foi efetuada seguindo a ordem de prioridade expressa na tabela da figura 10 onde a seta indica o sentido crescente de prioridade. O campo rastreador da notação para subsistema é empregado para evidenciar quais Atividades Essenciais foram alocadas ao subsistema básico;
3) Atividades Essenciais que, estando alocadas à classe que encapsula um Tipo de Entidade X, requisitam informações referentes a outro Tipo de Entidade Y são clientes de serviço fornecido pelo subsistema básico Gerente de Y (serviço de leitura).
4) Atividades Essenciais que, estando alocadas à classe que encapsula um Tipo de Entidade X, fornecem informações referentes a outro Tipo de Entidade Y são clientes de serviço fornecido pelo subsistema básico Gerente de Y (serviço de edição - inserção/remoção/alteração).

Esse mapeamento é a primeira etapa do processo de síntese. Refinamentos das classes que compõem um sistema dependem da semântica associada ao Esquema de Atividades, a qual pode estabelecer diferentes graus de coesão entre atividades.

Normalmente, o Modelo de Classes obtido na primeira etapa do processo de síntese necessitará ser refinado visando respeitar critérios de coesão funcional entre operações que atuam sobre estruturas de dados encapsuladas em classes distintas. Esse refinamento visa à alocação de atividades e agregação de informações de forma a:

- alocar a uma mesma classe as atividades cujos conjuntos de operações obedecem à dinâmica definida por uma única função de controle;
- agregar as classes correspondentes a Tipos de Entidade associados através de relacionamento com sentido único (acesso indireto ao Tipo de Entidade fraco)

- agregar as classes correspondentes a Tipos de Entidade associados através de um Tipo de Relacionamento com Atributos, entre si e com a classe correspondente à com a Porção Entidade desse tipo de relacionamento, quando as operações que atuarem sobre essas estruturas forem fortemente coesas;
- agregar as classes relacionadas ao tratamento de um único dispositivo de hardware (sensor/atuador monitorado/controlado pelo sistema);

As alocações e agregações devem ser realizadas visando aumentar o grau de reúso das classes resultantes, isto é, os elementos funcionais (passivos e ativos) que seriam sempre empregados em conjunto quando instanciados em um sistema devem constituir uma única classe.

O critério de reúso de classes determina uma revisão adicional das classes obtidas de forma a:

- desagregar classes para explicitar características não reusáveis de operações, isolando-as de operações reusáveis. Por exemplo, explicitar aspectos de controle em um subsistema básico cuja estrutura de informação reflita apenas estados relevantes do sistema específico, uma característica particular e não reusável em outros sistemas;
- eliminar possíveis redundâncias entre classes. Isto é, devem ser agregadas classes derivadas de tipos de entidade especializados. Por exemplo, ao tratar dispositivos iguais com funções diferentes (motor para movimentar na direção x e motor para movimentar na direção y), o processo de derivação exposto precedentemente determina a existência de duas classes distintas. No entanto, essas classes correspondem ao encapsulamento de uma mesma estrutura de informação associada a um mesmo conjunto de operações; portanto, ambas dizem respeito a uma única classe de dispositivo (motor). O design do sistema em questão possuirá duas instâncias do mesmo subsistema básico derivado dessa classe.

A implementação de classes através de subsistemas básicos baseia-se na análise da necessidade de concorrência entre os objetos que instanciam a classe. Quando essa concorrência não for necessária, uma classe é implementada como um subsistema básico que gerencia um conjunto de objetos pertencentes a essa classe. Caso contrário, cada subsistema básico equivale a um objeto que instancia a classe.

Após a determinação do conteúdo de cada **subsistema básico** (estrutura de informação e operações), deve ser definida a forma de ativação de cada uma das suas operações, através da interface que conecta o subsistema básico considerado aos demais subsistemas básicos e ao ambiente externo ao sistema. Essa interface é obtida analisando-se as conexões existentes entre as Atividades Essenciais do Modelo do Comportamento e destas com o ambiente externo. Essa análise permite a determinação das **portas** existentes em cada **subsistema básico**.

As portas de um subsistema básico devem atender às necessidades de comunicação entre atividades encapsuladas em classes distintas e às necessidades de comunicação entre cada subsistema básico e o ambiente externo.

Portas de comunicação com o ambiente externo resultam das conexões existentes entre as Atividades Essenciais encapsuladas pela classe e esse ambiente e da concorrência existente entre operações. Uma conexão entre duas Atividades Essenciais não encapsuladas em um mesma classe dá origem a uma requisição de serviço entre as classes às quais pertencem e à existência de uma Porta de Saída no subsistema origem e de uma Porta de Entrada no subsistema fim da conexão. O número de Portas e os tipos de mensagem alocados a cada Porta dependem das características de concorrência existente entre as Atividades Essenciais encapsuladas pelo subsistema básico, características essas já definidas nos Diagramas de Estado e Transição que especificam as

Atividades de Controle do Esquema de Atividades. As portas são agregadas em função da concorrência existente entre os serviços ativados por uma porta de entrada, de forma a:

- explicitar serviços concorrentes em portas diferentes;
- reunir requisições de serviço com características não concorrentes em uma mesma porta. Cada tipo de mensagem diferente em uma porta de entrada equivale a requisições de serviços diferentes, não concorrentes, ativados através dessa porta.

As classes obtidas a partir do Modelo do Comportamento são implementadas através de subsistemas básicos que incorporam as Atividades Essenciais do sistema. A esses subsistemas devem ser acrescentadas operações sobre as estruturas de informação características da alternativa de implementação adotada, tais como a leitura de valores associados a fluxos de dados contínuos através de interrupção ou pooling. Ao conjunto de subsistemas devem ser acrescentados subsistemas contendo apenas atividades não essenciais, tais como como as associadas à implementação da Interface Usuário Máquina - IUM - (entrada, interpretação de comando e apresentação de resultados). Os subsistemas básicos que implementam a IUM devem explicitar subsistemas que tratam dispositivos de interface com usuário e subsistemas básicos gerentes da interpretação de comandos. O acréscimo desses novos subsistemas determina outra revisão das definições dos subsistemas básicos já gerados e das suas definições de portas, verificando-se a necessidade de novas portas devido à incorporacão dos serviços característicos da implementação.

O sistema compõe-se de subsistemas básicos e seu reúso em outro sistema pode gerar a necessidade de acrescentar novas portas ou novos serviços através de portas já existentes. É importante notar que:

- em uma dada configuração de um sistema, é possível que nem todos os serviços de uma classe sejam solicitados. Neste caso, as portas associadas a um serviço não solicitado ficam inoperantes (não são conectadas a portas de saída de outros subsistemas);
- o acréscimo de novas portas de entrada (acréscimo de serviços concorrentes) ou de novos serviços não concorrentes a portas já existentes de uma dada classe aumenta seu escopo de atuação e caracteriza a potencialidade de reúso da classe.

Os subsistemas básicos obtidos a partir dos critérios acima expostos devem poder ser classificados em [Bustard88, NRL]:

- **responsáveis por objetos físicos** (ex.: dispositivos de hardware): normalmente, a um dispositivo de hardware é acoplado um subsistema básico, que o controla e o apresenta ao sistema com um conjunto de operações necessárias a seu uso (Dispositivo Inteligente);
- **responsáveis por estruturas de dados**: visam manter a filosofia de encapsulamento de dados, bem como obter homogeneidade na segmentação, já que um dispositivo de hardware pode ser considerado uma estrutura de dados encapsulada em uma peça de hardware;
- **responsáveis por funções do sistema**: funções relacionadas, não associadas diretamente a objetos físicos ou a estruturas de dados, devem ser reunidas em um subsistema básico;
- **responsáveis por relacionamento de funções**: estes são os subsistemas básicos que coordenam a execução,encapsulam atividades de controle. Essa coordenação é o que determina a forma de operação do sistema.

As duas últimas categorias são necessárias para aumentar o número de subsistemas básicos reusáveis. Em particular, um subsistema básico deve ser totalmente reusável ou não reusável. Durante a agregação/segmentação, quaisquer aspectos não reusáveis (devido ao fato de seus requisitos não atederem aos anseios de outros usuários), presentes nos serviços de um subsistema básico, devem ser encapsulados e associados a um ou mais subsistemas básicos não reusáveis.

Aspectos reusáveis que não necessitem ser sempre usados em conjunto, também devem ser separados em subsistemas básicos diferentes.

6.3.3. Identificando a Organização Hierárquica

Um subsistema básico tem como característica principal o fato de oferecer um conjunto coeso de Serviços atuando sobre uma estrutura de informação correspondendo a um número reduzido - quase sempre um - de depósitos contidos no Esquema das Atividades Essenciais. Esses serviços são logicamente reunidos e possuem somente uma finalidade específica; conseqüentemente, são também de fácil entendimento. Isto é, englobam ações associadas a **áreas de responsabilidade** bem definidas de uma particular solução para o sistema global. As **interfaces** desses conjuntos indicam como os subsistemas básicos cooperam para implementar essa solução. A partir dessas agregações, a estrutura dinâmica e estática do sistema pode ser descrita em termos dos subsistemas básicos. Entretanto, o entendimento global dessa estrutura é muito difícil no caso de sistemas de grande porte, onde será considerável o número de subsistemas básicos.

A organização dos **subsistemas básicos** em termos de uma estrutura hierárquica de subsistemas e serviços introduz novas agregações que facilitam a compreensão da estrutura global do sistema. A representação hierárquica agrega subsistemas (e seus serviços) em subsistemas (e serviços) de maior nível de abstração, em tantos níveis quantos forem necessários. A cada subsistema agregado, associa-se uma representação de serviços agregados alocados aos subsistemas que o compõe (um serviço agregado é uma representação abstrata que corresponde à união dos serviços alocados aos subsistemas componentes).

Para facilitar essa organização, é possível estabelecer que:

- subsistemas derivados das Atividades Essenciais devem ser agregados em um ramo da organização hierárquica separados dos subsistemas característicos da IUM. Dessa forma, o nível mais alto de abstração do sistema apresenta dois subsistemas: um subsistema que corresponde à parte funcional da aplicação e outro que incorpora a IUM. Deve ser estabelecido um protocolo de comunicação entre esses subsistemas;
- a hierarquia do subsistema que engloba a parte funcional da aplicação deve privilegiar a separação (em níveis altos de abstração) entre as atividades tipicamente operacionais e as atividades associadas a gerência de dispositivos (tratamento de hardware ligado a sensores/atuadores). Essa segmentação deve refletir uma agregação de dados e controle em subsistemas distintos daqueles que gerenciam o tratamento de dispositivos de hardware;
- subsistemas associados a gerência de dispositivos devem ser hierarquizados de forma a explicitar o relacionamento existente entre dispositivos. Por exemplo, devem ser agregados dispositivos com funcionamento interligado para atendimento de uma mesma necessidade do ambiente externo;
- a agregação deve respeitar o critério de encapsulamento de dados;
- cada subsistema agregado deve poder usar a classificação estabelecida para os subsistemas básicos.

Cabe ressaltar que aqui é apresentada uma filosofia de particionamento do sistema e das etapas do processo de desenvolvimento. No entanto, não é possível determinar um método que chegue à melhor solução ou a uma solução única. Um design reflete sempre o talento do projetista. Entretanto, os critérios descritos acima e a busca da solução mais simples possível devem nortear o trabalho. Ainda, critérios objetivos de avaliação da solução, envolvendo análise de acoplamento e coesão funcional, devem também ser usados visando medir da qualidade do trabalho realizado.

6.4. Especificando Subsistemas Básicos

No Projeto Básico, identifica-se o conjunto de serviços alocado a cada subsistema básico de uma forma ainda abstrata e sucinta. Entretanto, para que a implementação possa ser realizada, cada subsistema básico necessita de uma **Especificação Detalhada** independente. Isto é, os serviços identificados no Projeto Básico não estão especificados com o nível de detalhe necessário à implementação.

O Modelo da Essência contém a especificação de cada atividade primitiva em termos diagramas de estado-transição e de listas de pré e pós-condições das ações. Cada **subsistema básico** inclui o design da implementação de um conjunto de ações e restrições associadas a essas atividades primitivas, relacionadas aos serviços a ele alocados. A essas ações e restrições, serão acrescidas aquelas necessárias ao atendimento de serviços requisitados por outros subsistemas básicos.

Como o design do sistema corresponde a uma planta da implementação, as listas de pré e pós-condições associadas às atividades operacionais primitivas alocadas a um subsistema básico serão mapeadas em uma especificação procedimental. A esse conjunto de especificações procedimentais são acrescidas outras, referentes aos serviços característicos da opção de implementação encontrada no projeto básico, isto é, serviços que permitam a interação entre subsistemas básicos.

6.5. Configuração do Sistema

A **configuração do sistema** é constituída de duas etapas:

a) configuração do software
b) configuração de hardware

A **configuração do software** especifica o número de instâncias de cada subsistema básico e sua forma de conexão, respeitando o projeto básico e os requisitos de tolerância a falhas e faltas. As conexões existentes entre portas de subsistemas básicos podem ser dos seguintes tipos:

a) um para um
b) um para muitos
c) muitos para um

Os tipos de conexão **a** e **c**, para uma porta de saída, poderão ter resposta associada e o tipo **b** funciona como um **Broadcasting Seletivo**.

A **arquitetura de hardware** (número de processadores, capacidade de processamento/-armazenamento de cada processador e as características da rede que os interliga) é derivada de estudos de mapeamento da configuração de software em diversas arquiteturas de hardware, selecionando-se aquela que melhor se adapte às considerações de:

- carga máxima de processamento/armazenamento;
- desempenho requerido;
- tolerância a falhas e faltas;
- previsões de evolução do sistema;
- vida útil.

A **configuração de hardware** utiliza a arquitetura de hardware e define o mapeamento da configuração de software nessa arquitetura. Isto é, o mapeamento de cada instância de subsistema

básico na arquitetura de hardware (processador associado a cada processo de cada instância de subsistema básico) e a prioridade relativa entre processos alocados a um mesmo processador.

A **configuração do sistema** (configuração de software e configuração de hardware) especifica quais subsistemas básicos compõem o sistema, sua forma de cooperação e o mapeamento dos subsistemas básicos na arquitetura de hardware. Toda essa informação de configuração deve ficar isolada em uma ou mais unidades de implementação, diferentes daquelas que implementam os subsistemas básicos. Isto é, em caso de mudanças na configuração do sistema, as alterações deverão estar concentradas em códigos fonte independentes. Devido a essa independência e às características do software de aplicação provida pela técnica de design (subsistemas básicos autônomos), é possível reavaliar a configuração durante todo o processo de desenvolvimento sem incorrer em custos adicionais, corrigindo desta forma os erros de predição do desempenho do sistema, visando atingir de forma satisfatória os requisitos fixados pelo cliente/usuário.

7. Linguagens de Representação

7.1 - Diagramas de Estrutura

Os Diagramas de Estrutura (DE) modelam operações e dados. Apresentam de forma estática a decomposição de um sistema/subsistema em um conjunto de subsistemas. A um sistema/subsistema estão associados serviços, que presta/requisita ao ambiente exterior a ele, e portas de entrada/saída através das quais esse serviços são ativados. A integridade da estrutura de um sistema/subsistema representado por um DE é mantida exigindo-se que:

- um serviço esteja associado a um único subsistema;
- um serviço associado a um subsistema componente seja a totalidade ou fração de um serviço associado ao subsistema decomposto;
- serviços só sejam adicionados visando atender à comunicação entre subsistemas do DE; por exemplo, quando um subsistema necessita, para executar seu serviço, ler/atualizar dados da estrutura de informação de outro subsistema;
- portas e dados associados a um subsistema estejam presentes no DE que representa a decomposição do subsistema;
- portas e dados adicionais existam somente quando existir comunicação entre subsistemas componentes.

Diagramas de Estrutura são constituídos pelos seguintes elementos básicos:

- **subsistema**, representado por um retângulo com duas barra horizontais superior e inferior contendo, na parte central, o descritor do subsistema que deve resumir o conjunto de serviços por ele realizado, na parte superior, um identificador correspondendo ao nível em que se encontra na organização hierárquica e, na parte inferior, um campo reservado para comentários ou rastreamento;
- **fluxo de dados**, representado por uma linha (horizontal ou vertical) com setas, apresentando um descritor para os dados que fluem sempre próximo à seta que indica o sentido de deslocamento do dado. Modela a **interface** entre subsistemas;
- **porta,** representada por um pequeno retângulo inserido numa aresta vertical da notação para subsistema, contendo setas que indicam o tipo de porta e, portanto, a direção de deslocamento do dado que nela flui. A ordem vertical (horizontal) das setas indica indica a ordem da incidência e emergência dos dados, já que é possível ter fluxos bidirecionados em portas de entrada ou saída com resposta associada. A seta superior (da esquerda) indica o fluxo que ocorre primeiro. Portas agregadas mistas, geralmente presentes em níveis mais altos da

organização hierárquica, representam a agregação de portas de tipos diferentes e, portanto, devem ser representadas sem setas.

7.1.1. Exemplos de Representação

As figuras abaixo ilustram e complementam a sintaxe e semântica dos Diagramas de Estrutura, representando os elementos de modelagem e construções possíveis.

Elementos básicos de Modelagem

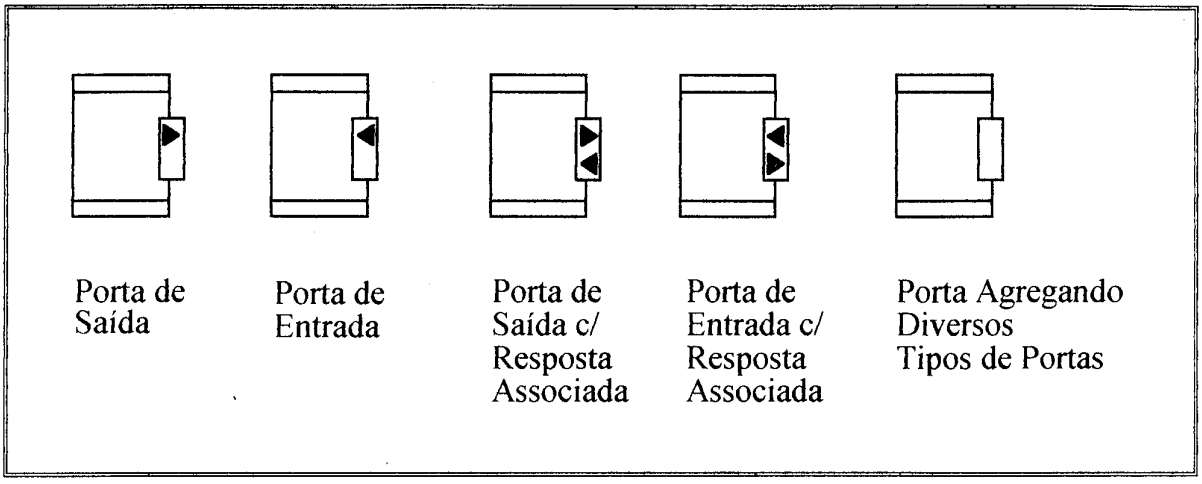
Subsistemas

identificador descritor comentário	• no centro, o descritor do subsistema, indicativo do conjunto de serviços alocados; • na parte superior, o identificador do subsistema, refletindo o nível ocupado na organização hierárquica de subsistemas; • na parte inferior, campo destinado a comentário ou rastreamento; em particular, pode ser usado para indicar existência de mais de uma instância de subsistema básico.

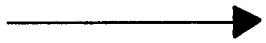
identificador descritor comentário	• o asterisco na parte superior direita indica que este é um subsistema básico.

identificador = Sn[{.m}]
n,m = natural

Portas



Conexões



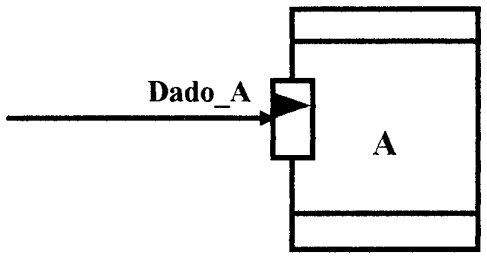
Conexão Unidirecional

- Representada por um fluxo de dados unidirecional, modela uma conexão causal;
- Conecta portas de entrada a portas de saída e vice-versa.

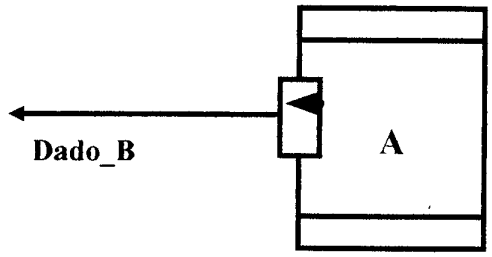
Conexão Bidirecional

- Representada por um fluxo de dados bidirecional, modela uma conexão causal e sincronismo de comunicação;
- Conecta portas de entrada com resposta associada a portas de saída com resposta associada e vice-versa.

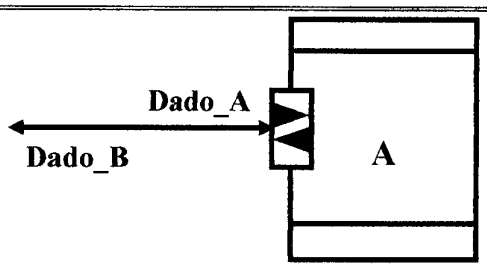
Composição dos Elementos de Modelagem



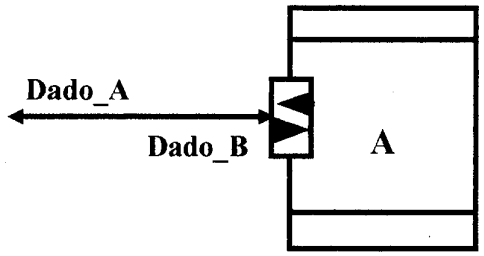
- O subsistema A recebe estímulo para executar um serviço (requisição de serviço) através de sua porta de entrada.
- A requisição de serviço contém o dado de entrada Dado_A, necessário à execução do Serviço.



- O subsistema A requisita um serviço de um outro subsistema através de sua porta de saída.
- A requisição de serviço contém o dado de saída Dado_B.
- Existe uma representação de causalidade já que a chegada do Dado_B pressupõe o início da execução de um serviço sem que haja estado de espera por parte do requisitante do serviço.

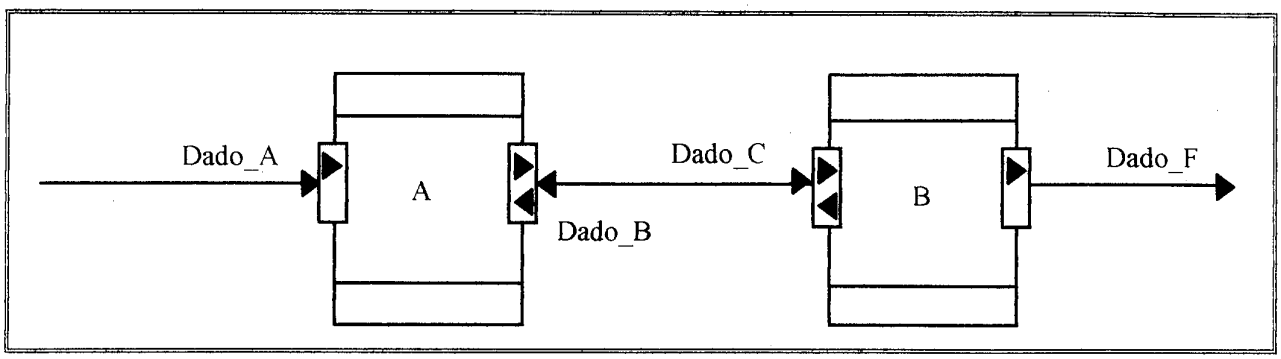


- O subsistema A recebe estímulo (requisição de serviço) através da porta de entrada com resposta associada.
- Esse requisição de serviço contém o dado de entrada Dado_A e a resposta o dado de saída Dado_B.



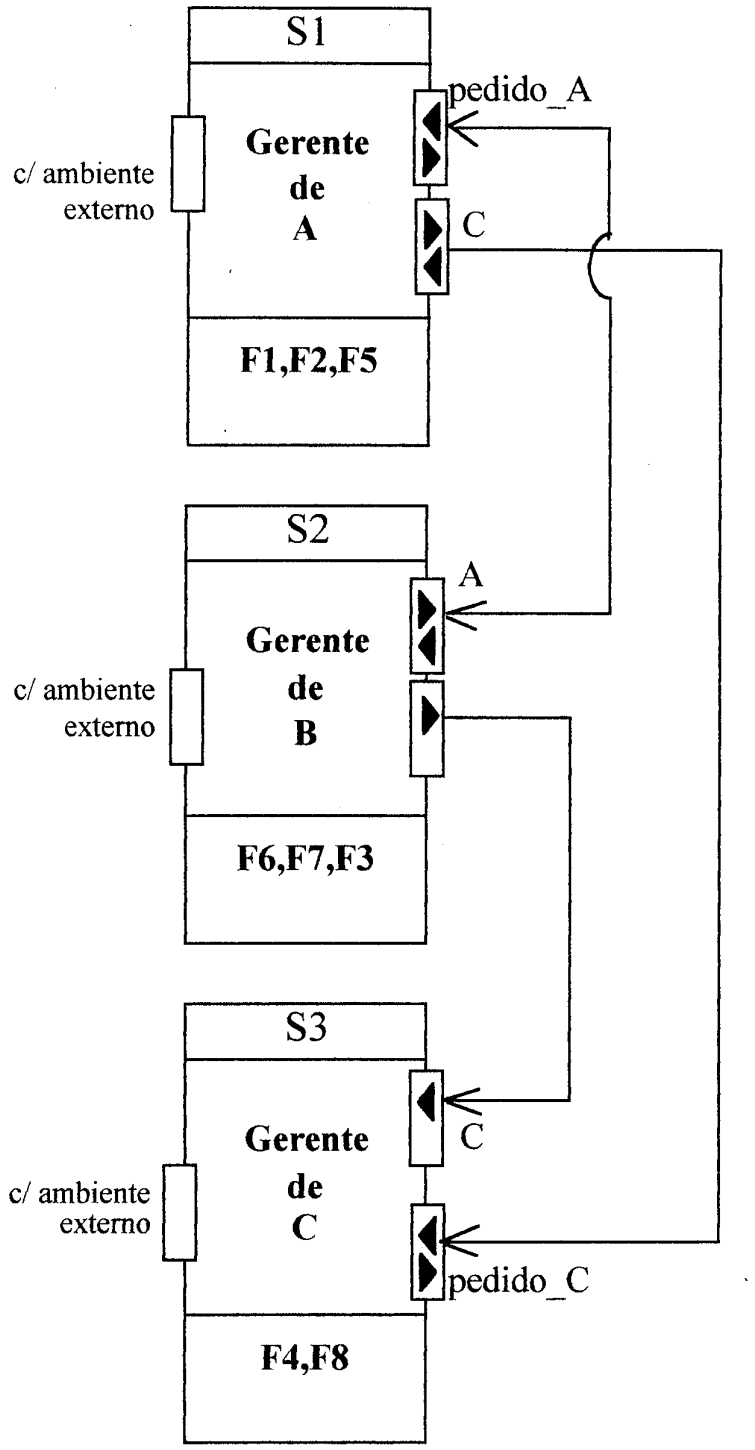
- O subsistema A requisita um serviço a ser executado por outro subsistema através de sua porta de saída com resposta associada.
- Esse requisição de serviço contém o dado de saída Dado_A e a resposta o dado de entrada Dado_B.
- O subsistema A pode prosseguir sua execução enquanto não necessitar do Dado_B.
- Ao estabelecer-se a necessidade do Dado_B, o subsistema A requisita-o à porta. Caso o Dado_B não esteja disponível na porta, o processo a ela associado fica suspenso até recebê-lo ou até que ocorra uma resposta na porta indicativa de falha/falta no sistema.

Construção Clássica



- A chegada do Dado_A equivale a uma requisição de serviço ao subsistema A o que implica iniciar a execução desse serviço.
- O subsistema_A, para completar a execução do serviço requisitado, necessita requisitar um serviço a outro subsistema através de sua porta de saída com resposta associada. Para requisição desse serviço, envia o Dado_C e prossegue sua execução até que necessite dessa resposta para poder continuar.
- Essa necessidade é expressa pelo subsistema através de uma requisição de mensagem na porta de saída com resposta associada. Caso a mensagem de resposta (Dado_C) já esteja disponível, o processo do subsistema A associado a essa porta recebe a resposta e continua sua execução. Caso não esteja, o processo é suspenso até recebimento da resposta (Dado_C) ou até recebimento de uma resposta que indique falha/falta no sistema.
- A porta de saída com resposta associada do subsistema A está conectada à porta de entrada com resposta associada do subsistema B.
- A chegada do Dado_B implica o início de execução do serviço por ele requisitado.
- Ao término do serviço, o subsistema B envia a resposta - Dado_B - para sua porta de entrada com resposta associada. A conexão que liga essa porta à porta de saída com resposta associada do subsistema A faz com que o Dado_B chegue ao subsistema A.
- Através de sua porta de saída, o subsistema B envia uma mensagem durante a execução do serviço requisitado através da porta de entrada com resposta associada ou durante a execução de um serviço interno do subsistema B concorrente com o primeiro. Nesse último caso, a execução desse serviço teria sido estimulada por um valor específico do tempo ou por ocorrência de condições limites (Serviço Interno do Susbsitema B).
- Subsistemas com serviços de execução periódica devem ter essa temporização indicada no campo de comentário.

Modelo de Subsistemas Básicos referente ao exemplo apresentado no item 6.3



7.1.2. Regras de Consistência

Cada Diagrama de Estrutura deve ser verificado segundo as seguintes regras de consistência:

- Todas as conexões associadas a um dado subsistema no nivel x devem aparecer na decomposição desse subsistema no nível x+1. Caso ocorra desagregação de fluxos, essa decomposição deve constar do Dicionário de Dados.

- Todo subsistema, exceto subsistema básico, deve ter sua estrutura descrita por um DE.
- Os subsistemas básicos devem possuir um "*", colocado no campo superior da notação, indicando que a eles não mais será associado um DE e sim uma especificação detalhada dos serviços a ele alocados.
- Todo subsistema deve produzir pelo menos uma informação de saida;
- Todo subsistema deve ter pelo menos um fluxo de entrada ou executar serviços internos (em decorrência de valor específico do tempo ou de uma condição limite).

Além dessas regras de consistência estrita é importante garantir que os DE's não possuem sintomas de "maus DE's", tais como:

- Número muito grande de interfaces entre dois subsistemas ou número muito grande de subsistemas. Isso revela, provavelmente, a falta de um nível intermediário ou nova divisão de responsabilidade associada aos serviços que o sistema deve executar.
- Repetição do mesmo serviço em mais de um subsistema, redundância que degrada a Manutenibilidade.
- Um número muito grande de níveis, revelando provavelmente um projeto básico excessivamente complexo em função do porte do sistema sendo projetado. O uso da recursividade inerente à técnica (correspondente à recursividade do conceito de sistema) permite a geração de um projeto básico preliminar, no qual o sistema é decomposto em subsistemas (não básicos) de menor porte. A seguir, cada subsistema será tratado como um sistema e a cada um deles associado um projeto básico.

7.2. Diagramas de Comportamento

Os Diagramas de Comportamento [Manual84] apresentam os serviços de cada subsistema e os Pontos de Controle que estabelecem a precedência/concorrência entre as Ações dos subsistemas (execução total ou parcial de um serviço). Um Ponto de Controle refere-se a um acontecimento num dado instante de tempo que possui a propriedade de ser memorizado, isto é, indica a saída de um estado que prevaleceu até aquele dado instante. Ao ser usado é consumido.

Os DC's modelam operações, dados e controle, através de diagramas com a seguinte simbologia:

- **Subsistema/Serviço/Ação**, representado por um círculo, dividido por uma barra horizontal. Na parte superior registra-se o identificador do subsistema seguido pelo identificador do serviço responsável pela Ação, que é descrita na parte inferior.
- **Ponto de Controle**, representado por uma Barra.
- **Dado**, representado por um descritor de fluxo de dado, associado a um Ponto de Controle. Significa que esse Ponto de Controle equivale à ocorrência da mensagem que transporta esse dado ou de um estado do subsistema, modelando dados produzidos internamente a ele.
- **Controle** representa uma regra de composição/decomposição de Pontos de Controle. Utiliza conectores análogos aos conectivos da Lógica. São eles:
  x - análogo a E
  * - análogo a OU inclusivo
  + - análogo a OU exclusivo

Na composição, dois ou mais Pontos de Controle são ligados por arcos direcionados à barra do Ponto de Controle composto. O Ponto de Controle composto só ocorrerá se aqueles que o compõem ocorrerem segundo a regra de composição. Na decomposição, um Ponto de Controle é ligado, por arcos direcionados de saída, a dois ou mais Pontos, nos quais se decompõe segundo suas regras de decomposição. Só é permitida uma regra de composição/decomposição por Ponto de Controle mas é possível o encadeamento dessas regras. Por exemplo, um Ponto de Controle se decompõe em

dois, segundo um E (a ocorrência dos dois é simultânea) e um desses se decompõe em outros dois, segundo um OU Exclusivo (ocorre apenas um dos dois últimos Pontos de Controle resultantes).

A execução de uma Ação de um subsistema representado no DC só acontece quando, a partir de sua regra de decomposição, ocorre o Ponto de Controle de entrada. O resultado desta execução dá origem ao Ponto de Controle de saída do Subsistema/Serviço/Ação considerado, decomposto segundo suas regras de decomposição. Essas regras de composição/decomposição dos Pontos de Controle, fazem parte do código responsável por implementar o serviço em questão, já que representam o seu padrão de comportamento.

Apesar de existir nos DC's a mistura de dados, operação e controle em um mesmo diagrama, sua construção e leitura são simplificadas pois trata-se de diagramas segmentados através da hierarquia correspondente aos diversos níveis de particionamento do sistema em subsistemas, apresentados nos DE's.

7.2.1. Exemplos de Representação

A seguir apresentamos diversas figuras que ilustram e complementam a exposição da seção precedente, representando todos os elementos de modelagem e as construções descritas.

Elementos básicos de Modelagem

	• Modela uma *Ação* executada por um subsistema, onde *Ação* pode ser um serviço ou parte dele.
	• Modela um *Ponto de Controle*. Junto a um Ponto de Controle pode constar um descritor de um dado presente no DE correspondente ou de um estado do subsistema, modelando dados produzidos internamente ao subsistema. • A ocorrência de um ponto de controle representa um evento que só é consumido após seu processamento por uma ação.

	• Modela uma *Conexão*, representando ligação entre dois dos elementos de modelagem acima mencionados.
	• Uma conexão pode unir:
	  • uma ação a um ponto de controle;
	  • um ponto de controle a uma ação;
	  • dois pontos de controle.

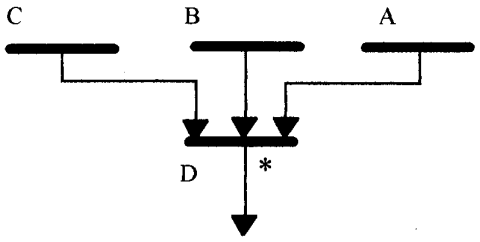
*	• Conector que modela uma operação de composição com valor análogo ao OU inclusivo da lógica.

x	• Conector que modela uma operação de composição com valor análogo ao E da lógica.

+	• Conector que modela uma operação de composição com valor análogo ao OU exclusivo da lógica.

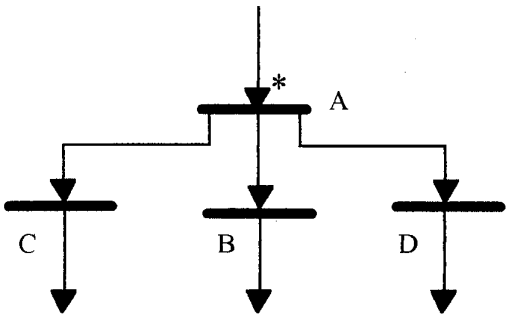
" "	• Comentário, muitas vezes necessário para aprimorar o entendimento do diagrama.

Composição de Pontos de Controle - OU Inclusivo



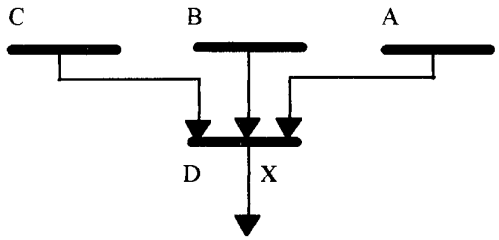
- A ocorrência de qualquer um dos três pontos de controle (A, B ou C), ou uma composição deles, equivale à ocorrência do ponto de controle D.
- Com esta notação representa-se a possibilidade de concorrência existente entre a ocorrência dos pontos de controle A, B e C.

Decomposição de Pontos de Controle - OU Inclusivo



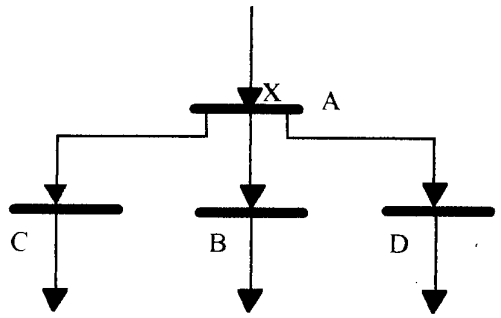
- A ocorrência do ponto de controle A equivale à ocorrência de um ou mais dos pontos de controle subseqüentes, isto é, pode dar origem por exemplo à ocorrência do ponto de controle B, ou aos pontos de controle C e B ou aos pontos de controle C, B e D.
- Com esta notação representa-se a possibilidade de concorrência existente entre a ocorrência dos pontos de controle A, B e C.

Composição de Pontos de Controle - E



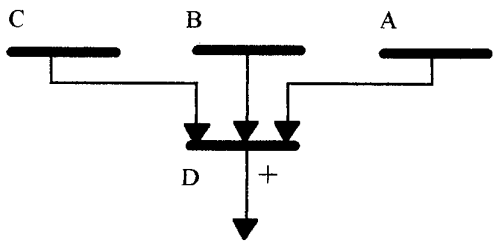
- Somente a ocorrência dos três pontos de controle (A, B e C) equivale à ocorrência do ponto de controle D.
- A ocorrência dos pontos de controle A, B e C não necessita ser simultânea; nesta notação está implícita a memória do sistema registrando a ocorrência de pontos de controle ainda não processados.

Decomposição de Pontos de Controle - E



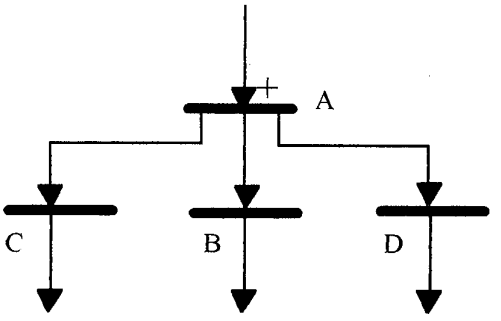
- A ocorrência do ponto de controle A equivale à ocorrência simultânea dos pontos de controle B, C e D.
- Nesta caso o sincronismo e a simultaneidade são especificados.

Composição de Pontos de Controle - OU exclusivo



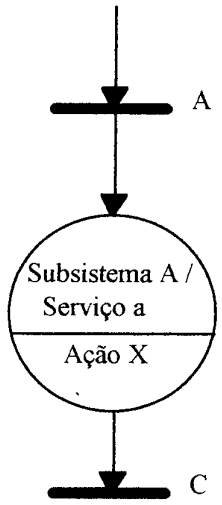
- A ocorrência exclusiva de um dos três pontos de controle (A, B ou C) equivale à ocorrência do ponto de controle D.
- Modela a seqüencialização do processamento dos pontos de controle pois, ao ocorrer um deles, o ponto de controle D estará habilitado, e disparará a ação a ele ligada. A ocorrência dos outros pontos só será atendida após término de execução da ação.

Decomposição de Pontos de Controle - OU exclusivo

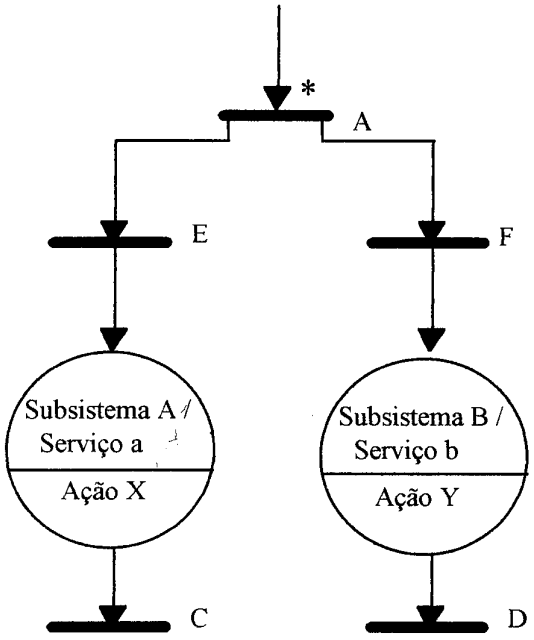


- A ocorrência do ponto de controle A equivale à ocorrência exclusiva de um dos pontos de controle subseqüentes (B, C ou D).
- Esta notaçao implica a não existência de concorrência entre as ocorrências desses pontos de controle.

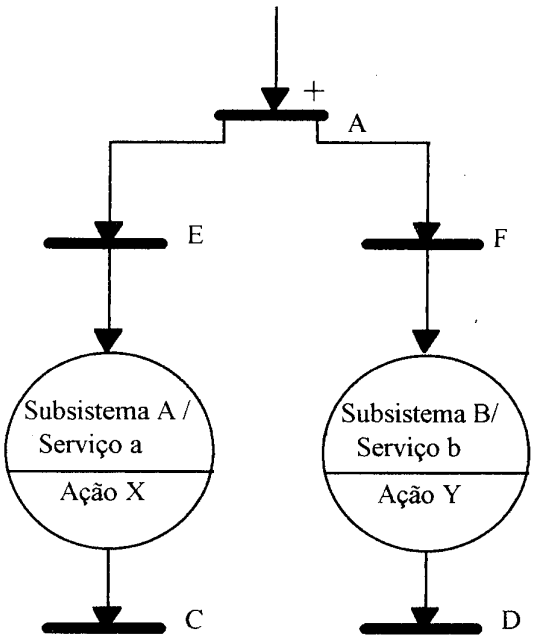
Ativação de Serviços em subsistemas



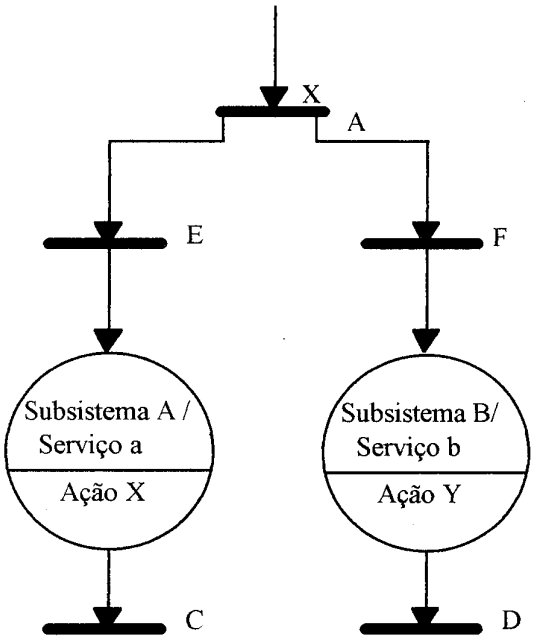
- A ocorrência do ponto de controle A causa a execução pelo subsistema A da Ação X.
- O témino da execução da Ação X dá origem ao ponto de controle C.



- A ocorrência do ponto de controle A equivale à ocorrência dos pontos de controle E e/ou F.
- No caso da ocorrência de ambos os pontos de controle, as Ações X e Y executam concorrentemente.
- Essa notação denota a possibilidade de existir a concorrência entre duas ou mais Ações de subsistemas (iguais ou diferentes).

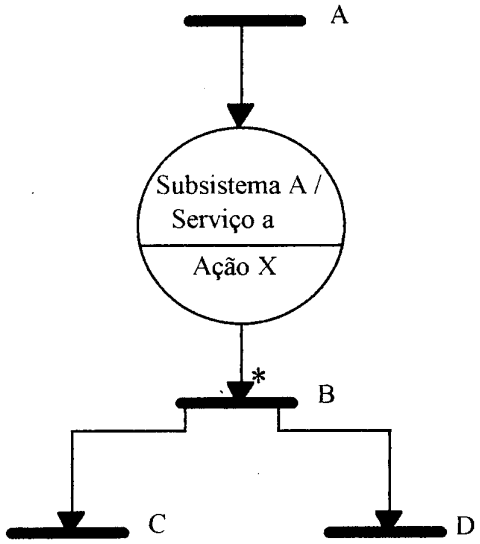


- A ocorrência do ponto de controle A equivale à ocorrência de um dos pontos de controle E ou F.
- Essa notação denota que não existe a possibilidade de concorrência entre duas ou mais Ações de subsistemas (iguais ou diferentes).

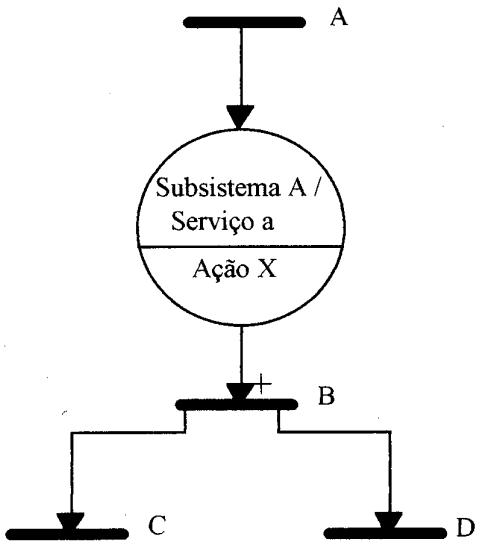


- A ocorrência do ponto de controle A equivale à ocorrência simultânea dos pontos de controle E e F.
- Essa notação denota que existe concorrência entre duas ou mais Ações de subsistemas (iguais ou diferentes).

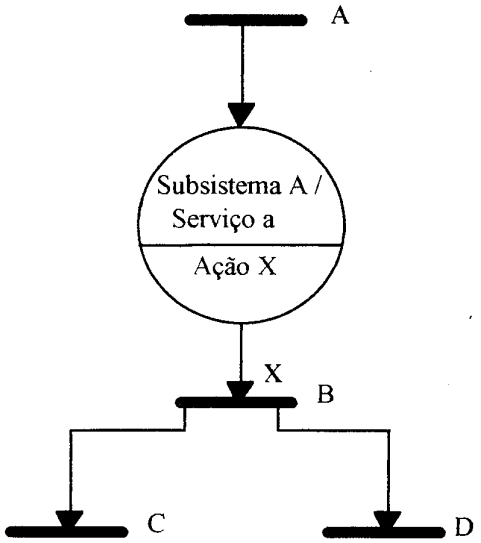
Geração de Pontos de Controle e sua Decomposição

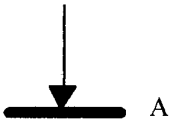


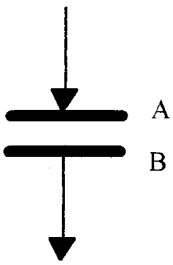
- A ocorrência do ponto de controle A causa a execução da Ação X do Subsistema A. Essa execução dá origem ao ponto de controle B que é equivalente à ocorrência dos pontos de controle C e/ou D.
- Pode ou não existir concorrência na ocorrência dos pontos de controle C e D.



- A ocorrência do ponto de controle A causa a execução da Ação X do Subsistema A. Essa execução dá origem ao ponto de controle B que é equivalente à ocorrência dos pontos de controle C ou D.
- Não existe concorrência na ocorrência dos pontos de controle C e D.

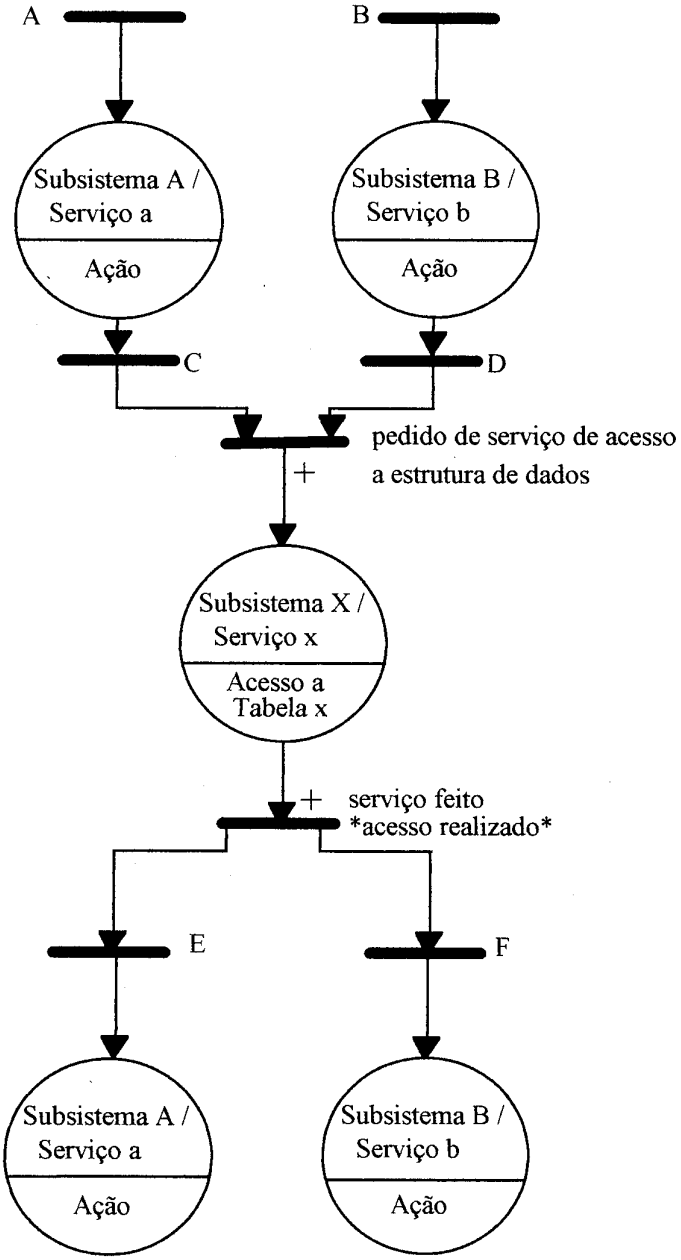
	• A ocorrência do ponto de controle A causa a execução da Ação X do Subsistema A. Essa execução dá origem ao ponto de controle B que é equivalente à ocorrência simuntânea dos pontos de controle C e D. • Existe concorrência na execução das Ações habilitadas pelos pontos de controle C e D.

	• A ocorrência do ponto de controle A é sinalizada para outro subsistema ou para Entidade Externa.

	• A ocorrência do ponto de controle A é sinalizada para outro subsistema que envia uma resposta sinalizada pela ocorrência do ponto de controle B.

	• Conectores empregados para indicação de continuação, visando melhorar o entendimento do Diagrama. • Contém uma numeração para identificação da outra ponta da conexão.

Construção Clássica
Acesso a Dados Compartilhados



7.2.2. Regras de Consistência

Todo DC deve ser verificado segundo as seguintes regras de consistência:

- A toda ação está associada um Ponto de Controle que a estimula para iniciar a execução. Este Ponto de Controle pode ser um fluxo de dados presente no DE associado, uma ocorrência de valor específico de Tempo (condição temporal) ou um Ponto de Controle equivalente à composição ou decomposição de outros Ponto de Controles.
- Após sua execução, toda ação dá origem a um Ponto de Controle, que pode ou não se decompor em outros Ponto de Controles. Se a execução der origem apenas a uma Resposta Interna, o descritor do Ponto de Controle deve mencioná-la (ponto de controle com descritor de "feito").
- Todos os fluxos de dados do DE associado devem aparecer no DC associado.
- Todos os subsistemas do DE , e apenas esses, devem aparecer no DC associado.

8. Estrutura do Projeto Básico

Para o Projeto Básico é usada uma linguagem de representação de estrutura (Diagramas de Estrutura - DE) apoiada pela linguagem de representação de controle (Diagramas de Comportamento - DC). Essas linguagens são hierárquicas, possibilitando a visualização da estrutura e da concorrência entre subsistemas. São modeladas **as Operações** executadas pelo sistema, através de seus subsistemas, e **os Dados** produzidos que trafegam nas mensagens. O Controle é modelado através do uso de DC's que evidenciam a concorrência entre os serviços prestados pelos diversos subsistemas que constituem um DE.

O registro das decisões de design é feito por pares (DE, DC), um para cada nível da estrutura de decomposição, desde a visão do contexto do sistema (neste caso sem DC) até níveis onde só existem subsistemas básicos. O uso de Diagrama de Comportamento associado a cada Diagrama de Estrutura permite um domínio eficiente de complexidade e facilita o entendimento global do comportamento do sistema. O conjunto de DE's e DC's proporciona uma visão hierárquica do sistema até o nível de subsistemas básicos. A cada DE fica associada uma lista dos serviços prestados, englobando todos os subsistemas componentes. Esta prática facilita as verificações de integridade no que diz respeito a:

- não redundância de serviços;
- completeza de serviços prestados pelos subsistemas componentes em relação ao subsistema decomposto.

Para facilitar a visualização da interligação entre subsistemas básicos (o que facilita a definição da configuração de sistema), é incorporado um DE representando todos os subsistemas básicos e suas conexões - **Diagrama dos Subsistemas Básicos**. A validação do projeto básico com relação ao Modelo da Essência é garantida através da inclusão, nos campos de rastreamento dos subsistemas básicos, as Atividades Essenciais a eles alocadas.

Para explicitar a estrutura hierárquica de decomposição, complementa-se o projeto básico com uma **Árvore de Subsistemas**. Um Dicionário de Dados define os **dados** e **pontos de controle** presentes nos diagramas, representativos do conteúdo das mensagens e de estados de execução respectivamente.

8.1. Organização do Projeto Básico

O Projeto Básico deve ser organizado da seguinte forma:

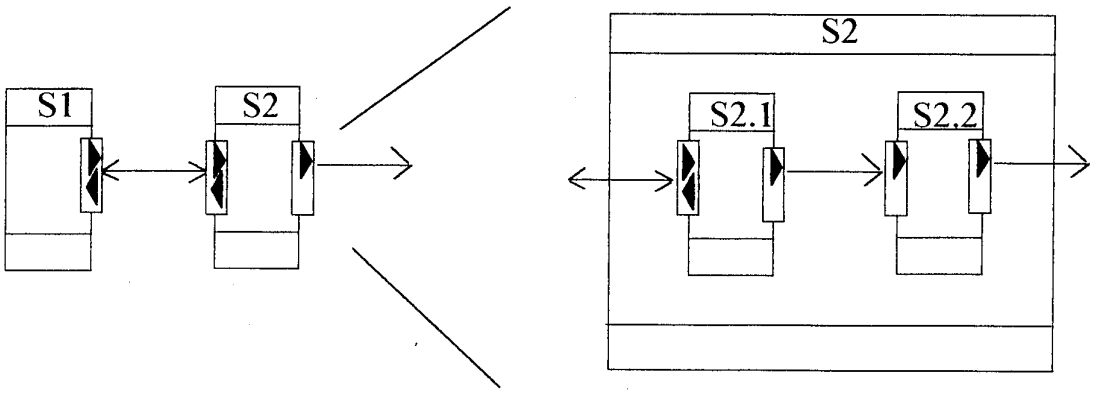
- **Contexto**
  Diagrama de Estrutura, representando o Contexto do Sistema;
  Descrição Textual, apresentando sucintamente cada serviço que compõe o sistema
- **Decomposição do Sistema**
  Apresentar na seguinte ordem, para cada nível de decomposição :
  Diagrama de Estrutura
  Diagrama de Comportamento
  Descrição textual, sucinta dos Serviços alocados a cada subsistema presente no Diagrama de Estrutura
- **Diagrama dos Subsistemas Básicos**
- **Árvore de Subsistemas**
- **Tabela de Verificação de Consistência com o Modelo da Essência**
  Tabela relacionando Serviços x Eventos Externos
  Tabela relacionando Atividades Essenciais x Subsistemas Básicos
- **Dicionário de Dados**

8.2. Descrição Textual Sucinta dos Serviços

Lista numerada, estruturada, associada a cada DE, que relaciona de forma compacta os serviços realizados pelo sistema/subsistema. Quando um serviço relacionado a um subsistema é agregado de serviços mais simples, essa decomposição deve aparecer sob a forma de estrutura do serviço associado ao subsistema sendo decomposto (vide figura 12). Essa decomposição se faz necessária para permitir a alocação de serviços ao novo nível de decomposição do sistema em subsistemas e a verificação de consistência entre a distribuição de serviços para os subsistemas presentes no diagrama.

Essa lista deve possuir o seguinte elemento estrutural:

n. Serviço abrangente
  {n.m. Detalhe do Serviço}
onde
n, m : natural seqüencial;
{} : indica iteração.



Descrição de Serviços
Subsistema S1
1. Serviço A
2. Serviço B

Subsistema S2
1. Serviço C
   1.1. Serviço C1
   1.2. Serviço C2

Descrição de Serviços
Subsistema S2.1
1. Serviço C1

Subsistema S2.2
1. Serviço C2

Figura 12 - Alocação de serviços a subsistemas

8.3. Diagrama de Subsistemas Básicos

Empregando a sintaxe e semântica dos Diagramas de Estrutura, esse diagrama apresenta apenas os subsistemas básicos e a forma como interagem e (portas, dados e conexões) no seu nível mais baixo de decomposição nesse escopo.

8.4. Árvore de Subsistemas

A Árvore de Subsistemas proporciona uma visão esquemática da estrutura hierárquica selecionada, facilitando a identificação da decomposição de um dado subsistema e auxiliando as atividades de manutenção. Nessa árvore os subsistemas básicos são as folhas.

8.5. Dicionário de Dados

Conjunto de dados presentes nos diagramas de estrutura e comportamento. Deve ser organizado em ordem alfabética e cada entrada no dicionário deve apresentar :

- **Nome**;
- **Definição**: define o significado do dado, interpretando o modelo em termos da realidade;
- **Composição**: em termos da BNF, especifica as relações entre os componentes de uma entrada estruturada;
- **Tipo**: especifica domínio (limites inferior e superior de um intervalo ou conjunto de valores), unidade, precisão e formato de uma entrada não estruturada (elemento de dado).

9. Subsistemas Básicos

9.1. Especificação Detalhada de um Subsistema Básico

A Especificação detalhada de cada subsistema básico é derivada da lista de pré- e pós-condições das Atividades Essenciais associadas a esse subsistema. As ações e restrições presentes nessas Atividades Essenciais são agora retratadas por uma linguagem algorítmica (linguagem estruturada), acrescidas de ações e restrições relativas a serviços prestados exclusivamente a outros subsistemas. Além da definição dos serviços (externos e internos) são também explicitados:

- serviços requisitados por todo e qualquer serviço executado pelo sistema;
- dados que definem as portas de comunicação desse subsistemas com outros;
- a estratégia de implementação do subsistema (por exemplo, a introdução de parâmetros que facilitem o reúso ou seleção de linguagem de programação), isto é, normas de implementação normalmente provenientes dos Requisitos de Concepção de Sistema.

Essa especificação visa permitir o desenvolvimento do subsistema por uma equipe independente, que não necessita ter conhecimento de todo o sistema, e facilitar o reúso, apresentando toda a definição necessária à seleção de um subsistema já desenvolvido para emprego em novos sistemas.

Deve constar de:

- **Diagrama de Contexto do Subistema Básico**, representado através de um Diagrama de Estrutura que contém apenas o subsistema, suas portas e os dados que fluem por essas portas;
- **Descrição dos Serviços**, utilizando linguagem estruturada, especificam-se os **serviços externos** (prestados a outros subsistemas) e os **serviços internos** ressaltando suas fontes de ativação e o protocolo de comunicação através das portas de entrada/saída relacionadas com a execução de cada serviço;
- **Serviços Requisitados**: serviços externos providos por outros subsistemas cuja requisição deve ser feita pelo subsistema que está sendo especificado, durante a execução de um dos seus serviços. São mencionados: protocolos de requisição do serviço, resposta esperada e serviço externo do subsistema que requisita. Esse item visa facilitar o reúso do subsistema básico pois permite ao projetista identificar se na configuração do sistema onde será empregado esse subsistema, existem subsistemas prestadores de serviços equivalentes aos requisitados, segundo o protocolo esperado para todos os serviços externos ativos (portas de entrada conectadas) desse subsistema;
- **Descrição Detalhada dos Dados**: especificação das estruturas de encapsuladas pelo subsistema básico e pelas suas portas de entrada/saída. Opcionalmente, pode-se empregar a definição de tipos, que facilitam e padronizam a definição dos dados. Deve ser segmentada em: *definições de tipos*; *definições das estruturas de dados encapsuladas*, explicitando os tipos de dados encapsulados pelo subsistema básico em questão; *definição das mensagens*, explicitando os tipos de dados carregados pelas mensagens que ficam na interface do subsistema básico;
- **Requisitos de Desempenho**, como o tempo máximo ou médio para execução de um serviço;
- **Parâmetros do Subsistema Básico**, onde são especificadas em detalhe todas as variáveis do subsistema passíveis de parametrização; por exemplo, devem ser indicados: números máximos para armazenamento de um dado, alocação de memória etc.;
- **Subsídios de Implementação**: indicando as normas, utilitários, linguagem de programação, ambiente de desenvolvimento (por exemplo: compilador, emulador) ou procedimentos especiais na manipulação de dados, como a escolha de algum algoritmo específico.

9.2. Design de um Subsistema Básico

Um subsistema básico é formado de uma ou mais unidades de execução concorrentes - **processos** - gerenciados por um supervisor de tempo real. A determinação dos processos é o primeiro passo no design de um subsistema básico. A cada **serviço concorrente** executado pelo subsistema básico, serviço externo associado a uma porta de entrada ou serviço interno, está associado pelo menos um **processo**.

Na comunicação entre processos de subsistemas básicos diferentes é utilizado o conceito de **interface**. Uma interface é constituída de portas e canais de comunicação que as conectam, visando à comunicação entre processos. Comunicação e sincronismo são feitos através da troca de mensagens.

Considerações de desempenho podem determinar uma implementação do subsistema básico com áreas de dados compartilhadas e a utilização de recursos tradicionais de programação concorrente como monitores, semáforos e eventos além da troca menos estruturada de mensagens. O método apresentado não proíbe essa prática mas desaconselha pois:

- essa opção de design é menos manutenível, em função da necessária complexidade do código fonte gerado;
- faz com que o subsistema básico não seja portátil pois sua configuração no hardware não estará plenamente definida a partir da linguagem de configuração.

Deve-se evitar associar a serviços diferentes serviços que não sejam efetivamente concorrentes. Por exemplo, serviços que criam, cancelam ou alteram valores de um estrutura particular de dados não são na sua essência concorrentes pois, a execução de um destes serviços implica o bloqueio da estrutura de dados, o que serializa esses procedimentos. Tratá-los em conjunto, como um único serviço associado a uma única porta de entrada, evita dispêndio adicional ("overhead") com a necessidade de introduzir mecanismos de exclusão mútua durante o design do subsistema básico ao qual estejam incorporados.

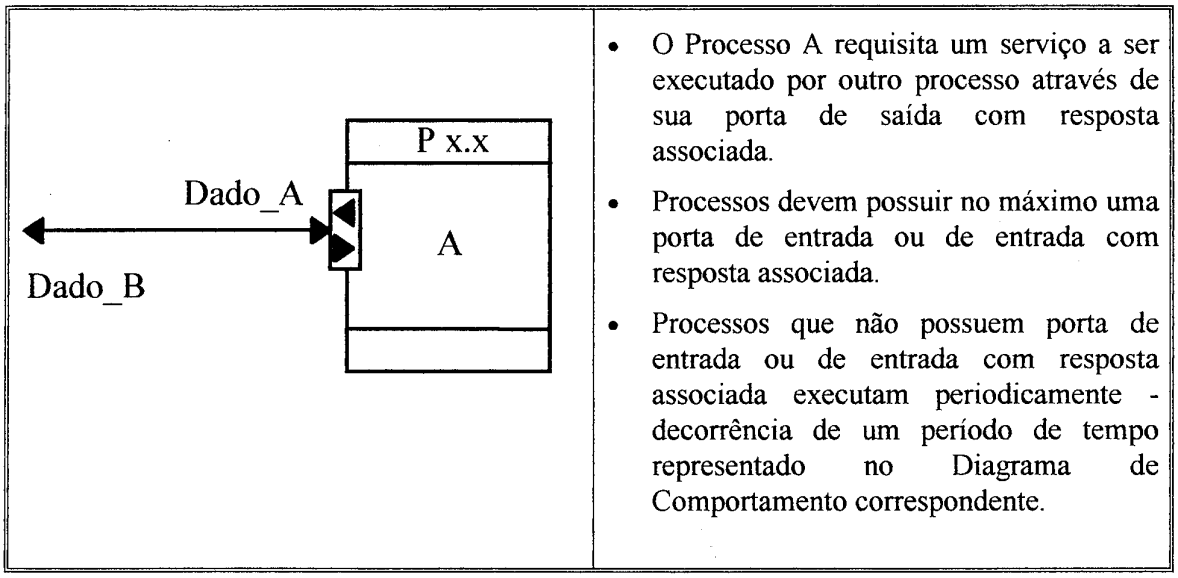
Um processo pode necessitar, para sua implementação, de um trabalho ainda grande de design pois, apesar de estar associado a serviços de fácil e rápido entendimento, tem que executar um conjunto de algoritmos, condições e controles, nem sempre triviais. É usada a decomposição tradicional em módulos ou rotinas, baseada em decomposição funcional, procurando-se as que possam ser vistas como **caixas pretas**, concentrando-se funções similares em uma mesma rotina e evitando-se o uso de variáveis globais. A decomposição funcional "top-down" é ainda um bom método para determinação das rotinas que compõem o processo. Para comunicação entre rotinas, já que aqui cada segmento tem natureza seqüencial, é usado o mecanismo tradicional de Call (chamada de sub-rotina).

O design de um subsistema básico deve constar de:

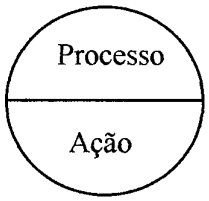
- **Diagrama de Estrutura**, utiliza a linguagem de representação de Diagramas de Estrutura e corresponde aos processos que constituem o subsistema básico;
- **Diagrama de Comportamento**, utiliza a linguagem de representação de Diagramas de Comportamento e corresponde à interação existente entre esses processos;
- **Especificação de cada processo**, correspondente ao serviço prestado pelo processo e, caso necessário, acrescenta-se à especificação as instruções que complementam a comunicação entre processos (texto em linguagem estruturada onde fica enfatizado em negrito o texto acrescido à especificação do serviço implementado pelo processo);

- **Descrição dos dados introduzidos**, que apresenta os dados introduzidos devido à necessidade de comunicação entre processos;
- **Estrutura interna de processos**, caso o processo seja suficientemente complexo, representado a estrutura hierárquica de módulos (árvore de módulos), complementada pela definição do objetivo de cada módulo. A especificação detalhada de cada módulo já consta da especificação do serviço que o processo implementa.

A especificação dos módulos dos processos não foi considerada pois normalmente chegam-se a módulos muito simples e sua especificação detalhada leva a uma quase codificação. Para módulos complexos, complementa-se o seu objetivo com uma especificação de funcionamento em linguagem estruturada. Essa especificação não necessita fazer parte do documento de design do sistema devido a grande coincidência entre ela e o código fonte, fator que reduz sua eficácia para entendimento futuro do módulo em tempo de manutenção. Ressalta-se o fato de que em manutenção ou teste essa especificação fica desatualizada, pelos motivos acima citados, sendo portanto uma fonte de inconsistências para o modelo.



- O Processo A requisita um serviço a ser executado por outro processo através de sua porta de saída com resposta associada.
- Processos devem possuir no máximo uma porta de entrada ou de entrada com resposta associada.
- Processos que não possuem porta de entrada ou de entrada com resposta associada executam periodicamente - decorrência de um período de tempo representado no Diagrama de Comportamento correspondente.



- No Diagrama de Comportamento a referência a subsistema é substituída pela referência a Processo

10. Serviços da Interface de Compatibilização

A implementação dessa camada é dependente apenas do software básico e do hardware sobre o qual é implementada, podendo ser reusada para qualquer aplicação que execute nesse mesmo ambiente.

Os serviços oferecidos, pela Interface de Compatibilização para a camada de aplicação, são bem definidos. Em caso de migração da aplicação para outro ambiente de software básico e/ou hardware, esta não necessita sofrer alterações, as alterações a serem efetuadas concentram-se apenas na Interface de Compatibilização.

A existência dessa interface evita limitações em termos de ambiente de desenvolvimento onde será desenvolvida a aplicação. A única limitação existente está no emprego para design da Aplicação dos conceitos da técnica apresentada nesse trabalho.

Os serviços a serem oferecidos pela Interface de Compatibilização, que suprem as necessidades do método de design proposto, são:

- **Serviços de Configuração de Software**, que implementam a definição de portas e interfaces.
- **Serviços de Configuração de Hardware**, que implementam a definição de processos, sua prioridades relativas e seu mapeamento na arquitetura de hardware.
- **Serviços de Comunicação**, que implementam o transporte de mensagens através de interfaces definidas entre subsistemas básicos.

Para que a linguagem de configuração possa verificar a consistência de portas conectadas através de interfaces, é necessária a definição dos tipos de dados associados a cada porta. Essa compatibilização de tipos seguirá a implementação de compatibilização de tipos (por nome ou estrutura) empregada pela linguagem de programação utilizada para seu desenvolvimento.

Uma porta de um processo de um subsistema básico é identificada por (nome da porta, nome do processo, nome da instância do subsistema básico). A porta é caracterizada pelo seu tipo (entrada, saída, entrada com resposta associada ou saída com resposta associada), o tipo de dados que por ela são enviados e/ou recebidos. Devido às características de enfileiramento de mensagens nas portas de entrada ou de entrada com resposta associada, a cada porta desse tipo deverá ser definido o tamanho máximo previsto para essa fila. Essa fila é implementada como uma lista circular cujo tamanho depende da relação produtor/consumidor existente entre subsistemas básicos conectados.

Para definição de interfaces deverá ser suprida um serviço de conexão de portas. No ato dessa conexão é verificado a consistência entre tipos de portas e tipos de dados das portas conectadas. As conexões serão sempre definidas no sentido da porta de entrada de um subistema básico com a porta de saída de outro subsistema básico a ela associada. São possíveis conexões entre:

- uma porta de entrada a uma ou mais portas de saída;
- uma porta de saída a várias portas de entrada (Broadcast Seletivo);
- uma porta de entrada com resposta associada a uma ou mais portas de saida com resposta associada;
- porta de entrada com resposta associada a uma ou mais portas de saída.

Neste último caso, a interface de compatibilização ao tentar enviar a resposta a porta de saída que requisitou a execução do serviço, despreza a mensagem contendo a resposta.

Como é possível conectar mais de uma porta de saida com resposta associada a uma única porta de entrada com resposta associada, é necessário, pela interface de compatibilização, o armazenamento, junto a cada mensagem em uma porta de entrada, da identificação da porta remetente da mensagem.

Como a preocupação desse trabalho é com sistemas concorrentes em ambiente distribuído possuindo normalmente requisitos de tolerância a falhas e faltas, é necessário que os serviços de comunicação permitam o tratamento de:

- falta ocorrida na rede de comunicação de dados;
- falta de um processador;
- falha que leve a um processo ficar suspenso por tempo indeterminado.

O tratamento dessas falhas e faltas não pode ficar totalmente sob a responsabilidade interface de compatibilização pois apenas a aplicação, implementada comi subsistema básico, pode identificar a relevância do problema ocorrido, especificando como tratá-lo (mensagens de alerta, alternativas de funcionamento ou funcionamento degradado).

O tratamento dessas ocorrências incluem também características de configuração de software e de hardware. Para continuar garantindo a independência dessas configurações da implementação do subsistema básico, a implementação desse tratamento é divido em:

- parte dependente da configuração, como a revisão da configuração de sistema (alteração de conexões entre portas), de responsabilidade de uma **rotina de recuperação**, ativada pela Interface de Compatibilização e integrante da configuração do sistema;
- parte independente da configuração, como envio de mensagens de erro e desativação de serviços, de responsabilidade do subsistema básico.

Associado a cada porta, a Interface de Compatibilização permitirá definir:

- tempo máximo de espera associado a porta;
- rotina de recuperação a ser ativada em caso de exceder o tempo máximo de espera.

Quando a uma porta de entrada (ou saída com resposta associada) for associado um tempo máximo de espera, a cada requisição de recebimento de mensagem na porta, além de suspender o processo, é iniciada uma contagem de tempo, pela Interface de Compatibilização. Caso essa contagem exceda o tempo máximo especificado e nenhuma mensagem seja recebida, a Interface de Compatibilização ativa a execução da rotina de recupeção associada a porta. Ao término de sua execução, o processo suspenso é liberado, com a ocorrência de uma mensagem na porta cujo conteúdo é originário da rotina de recuperação e com a informação do erro ocorrido.

Quando a uma porta de saída (ou saída com resposta associada) for associado um tempo máximo de espera, a cada requisição de envio de mensagem na porta, o processo é suspenso até que a Interface de Compatibilização consiga efetivar o envio da mensagem e a contagem de tempo é iniciada. Caso essa contagem exceda o tempo máximo especificado e a mensagem não esteja enviada, a Interface de Compatibilização ativa a execução da rotina de recupeção associada a porta. Ao término de sua execução, o processo suspenso é liberado, com a ocorrência de uma mensagem na porta cujo conteúdo é originário da rotina de recuperação e com a informação do erro ocorrido.

Ficará a cargo da **rotina de recuperação**:

- alterar conexão de portas, buscando ativar novo conectar a porta com a de uma outra instância do subsistema básico do qual não recebeu resposta ou para o qual nao conseguiu enviar mensagem;
- ativar uma nova instância de subsistema básico, provavelmente em outro processador, de forma a suprir o serviço que não conseguiu acessar;
- simular uma resposta ao subsistema básico contendo informações de erro (recuperável ou irrecuperável).

Ficará a cargo do subsistema básico, ao receber da rotina de recuperação a mensagem de **erro recuperável**, repetir o processo de envio/requisção de mensagem na porta. Ficará a cargo do subsistema básico, ao receber da rotina de recuperação a mensagem de **erro irrecuperável**:

- desativar o serviço externo que necessita acessar a porta cuja conexão está com problema ou abortar a execução do sistema, conforme a importância do serviço;
- enviar mensagens de alerta/erro explicativa aos responsáveis pela operação do sistema.

Essas funções de recuperação do sistema previstas na rotina de recuperação dependem de serviços de configuração dinâmicos providos pela Interface de Compatibilização. Como o intuito desse trabalho é definir técnica de design, os serviços de configuração de software e de hardware aqui definidos são empregados apenas na fase de inicialização do sistema (configuração estática do sistema). Durante a execução só poderão ser usados os serviços de comunicação. Por razões de simplificação da implementação dessa interface de compatibilização não está sendo prevista a existência uma reconfiguração de software e hardware dinâmica do sistema, a introdução desses serviços não altera em nada os conceitos de design expostos. Maiores detalhes sobre serviços desse tipo podem ser obtidos em [Magge94, Magge90].

10.1. Serviços de Configuração de Software

a) Serviço de Definição de Porta: Função Booleana Define_porta (porta_a, Tipo, Tam_msg, Tipo_msg, Tam_fila, Tempo_máximo, Rotina_recuperação)

Onde:

- ***porta_a***: identificação da porta da forma (Nome da porta.Nome do processo.Nome da instância do subsistema básico);
- ***Tipo*** : Entrada | Saída | Entrada com Resposta | Saída com resposta;
- ***Tam_msg***: Número de bytes do tipo que descreve a porta;
- ***Tipo_msg***: Nome do tipo empregado na definição da mensagem pela implementação do subsistema básico;
- ***Tam_fila***: Tamanho da fila de mensagens disponível para aquela porta;
- ***Tempo_máximo***: Tempo máximo de espera para o processo associado a essa porta, quando através dela, o processo ficar suspenso ao enviar/requisitar mensagens (parâmetro opcional).
- ***Rotina_recuperação***: rotina que tratará erro ocorrido na comunicação entre portas (parâmetro opcional apenas se a definição das portas envolvidas na conexão omite parâmetro relativo a tempo máximo de espera);
- ao definir uma porta, o subsistema básico eo o processo aos quais pertence já devem ter sido definidos. Retorna condição de errro caso não tenham sido definidos.

b) Serviço de Definição de Interfaces: Função Booleana Conecta (porta_a, porta_b)

Onde:

- ***porta_a***: porta de entrada ou de entrada com resposta associada;
- ***porta_b***: porta de saída ou de saída com resposta associada;
- As portas devem ser identificadas por (Nome da porta.Nome do processo.Nome da instância do subsistema básico);
- retorna condição de erro caso os tipos de portas conectadas não sejam compatíveis.

c) Serviço de Definição de Subsistema Básico: Função Booleana Define_SB (Nome, Instânciass)

Onde:

- ***Nome***: nome lógico que será associado ao subsistema básico;
- ***Instâncias***: número de instâncias de cada processo assciado ao subsistema básico que devem ser criados, isto é, o número de processos que executam o mesmo código;
- O nome lógico com o qual os subsistemas básicos que possuem mais de uma instância serão referenciados será o nome lógico do subsistema básico concatenado com números que vão de 1 até o número de encarnações requisitado;
- retorna condição de erro caso seja empregado o mesmo nome lógico para mais de um subsistema básico.

d) Serviço de Definição de Processo: Função Booleana Define_Processo (Nome, Procedimento, Ciclo, Nome_SB)

Onde:

- ***Nome***: nome lógico que será associado ao processo;
- ***Procedimento***: nome do procedimento que corresponde ao "entry point" para o início do módulo principal que implementa o processo;
- ***Ciclo***: parâmetro opcional que determina a periodicidade de processos cíclicos;
- ***Nome_SB***: nome lógico do subsistema básico ao qual pertence o processo;
- são criadas tantas instâncias do processo quanto o número de instâncias do subsistema básici ao qual pertencem;
- o nome lógico com o qual processos de subsistemas básicos que possuem mais de uma instância serão referenciados será o nome lógico do processo concatenado com números que vão de 1 até o número de instâncias do subsistema básico;
- retorna condição de erro caso seja empregado o mesmo nome lógico para mais de um processo ou caso o subsistema básico ao qual pertence ainda não esteja definido.

10.2. Serviços de Configuração de Hardware

a) Serviço de Mapeamento de Processos: Função Booleana Mapeia_Processo (Nome, Nó)

Onde:

- ***Nome***: nome lógico associado ao processo;
- ***Nó***: endereço do nó na configuração de hardware onde residirá o processo;
- retorna condição de erro caso o processo não esteja definido.

b) Serviço de Ativação de Processos: Ativa_Processo (Nome, Prioridade)

Onde:

- ***Nome***: nome lógico associado ao processo;
- ***Prioridade***: prioridade de execução do processo;
- o processo apesar de ser definido na configuração de software só é tornado uma unidade ativa de execução após o uso dessa primitiva;
- retorna condição de erro caso o processo não esteja definido.

10.3. Serviços de Comunicação

a) Serviço de Envio de Mensagens: Função Booleana Envia (porta_a, Msg, opção)

- **porta_a**: identificação da porta do subsistema básico que chama a primitiva de comunicação.
- **Msg**: endereço com o conteúdo da mensagem a enviar.
- **opção**: determina, caso não exista lugar na fila de mensagens para armazenamento da mensagem enviada, se o processo que deseja enviar a mensagem é suspenso ou a mensagem é perdida.

Essa primitiva determina o nome da encarnação da isntância do subsistema básico que identifica univocamente a porta. Através da definição de tipo de porta e das conexões, determina:

- caso a mensagem seja enviada a porta de saída (ou saida com resposta associada), com qual porta de entrada (ou entrada com resposta associada) está conectada e depositando a mensagem e seu remetente na fila associada a essa porta de entrada;
- caso a mensagem seja enviada a porta de entrada com resposta associada, determina a porta remetente da mensagem (aquela que ativou o serviço) para depositar nesta porta a resposta enviada;

Verifica a execução do envio de mensagens e:

- retorna condição de erro, caso o processo tente enviar mensagens a portas de entrada.
- detecta erro na transmissão de mensagens passando o controle à rotina de recuperação associada a porta_a. Retorna mensagem ao processo e condição de erro após término de execução da rotina de tratamento de recuperação.

b) Serviço de Recebimento de Mensagens: Função Booleana Recebe(porta_a, Msg, tam)

- **porta_a**: identificação da porta do subsistema básico que chama a primitiva de comunicação.
- **Msg**: endereço com a posição onde deve ser armazenado conteúdo da mensagem a receber.
- **tam**: tamanho em bytes da mensagem recebida.

Essa primitiva determina:

- a primeira mensagem na fila de mensagens da porta de entrada ou entrada com resposta associada de onde se deseja receber mensagens;
- a suspensão do processo, caso não exista mensagem na porta para ser recebida;
- a mensagem resposta recebida em uma porta de saída com resposta associada.

Verifica a execução do recebimento de mensagens e:

- retorna condição de erro, caso o processo tente receber mensagens de portas de saída.
- detecta erro na recepção de mensagens (excedido tempo máximo de espera por mensagem na porta) passando nesse caso o controle à rotina de recuperação associada a porta_a. Retorna mensagem ao processo e condição de erro após término de execução da rotina de tratamento de recuperação.

c) Serviço de Consulta de Mensagens: Consulta (porta_a, num_msg)

- **porta_a**: identificação da porta do subsistema básico
- **num_mg**: número de mensagens existentes na porta esperando para serem processadas.

Essa primitiva se faz necessária para permitir a interrupção de um processamento antes de seu final normal.

10.4. Documentação da Configuração do Sistema

O Projeto Básico já registra a configuração de software no Diagrama de Subsistemas Básicos, onde estão explicitados todos os subsistemas básicos, sua portas e conexões. O design de um subsistema básico especifica os processos, as portas e as conexões entre portas dos processos que constituem o subsistema básico.

É ainda necessário explicitar a configuração de hardware, isto é, a arquitetura de hardware selecionada e o mapeamento dos processos de cada subsistema básico nessa arquitetura. Visando respeitar a independência existente entre os modelos de configração de software e configuração de hardware, este último deve constituir um documento independente do projeto básico. Sua estrutura e conteúdo devem corresponder a:

- Diagrama de Blocos definindo arquitetura de Hardware e nomeando os processadores;
- Estudo de desempenho que valide a escolha dessa arquitetura (considerando-se os tempos de resposta médio e com carga de processamento máxima) frente aos requisitos de concepção do sistema;
- Diagrama Geral de Processos, apresentando um Diagrama de Estrutura contendo todos os processos de todos os subsistemas básicos, suas portas e conexões, evidenciando-se no seu campo rastreador a identificação do processador onde executa e a instância do subsistema básico ao qual pertence.

11. Emprego do Método

11.1. Implantação do Reúso

Após o desenvolvimento de alguns sistemas segundo o método aqui apresentado, a organização disporá de uma biblioteca significativa de subsistemas básicos reusáveis. Durante a concepção do projeto básico para um sistema novo, deve ser feita uma pesquisa nessa biblioteca, para determinar os subsistemas básicos existentes cujas características, obtidas a partir de suas especificações, sugiram reúso. A decomposição do sistema em subsistemas deve ser feita com esse conhecimento prévio e direcionada a aproveitar ao máximo subsistemas básicos existentes. Um sistema pode até vir a ser construído totalmente pela conexão de subsistemas básicos já desenvolvidos, como hoje em dia se constrói um módulo de hardware pela interligação de componentes integrados disponíveis no mercado.

Para a implantação da política de reúso, é necessário que o método de desenvolvimento esteja integrado a uma política administrativa que:

- incentive o reúso de subsistemas básicos, pois programadores são inclinados a produzir suas próprias soluções ao invés de usar as já existente [Panel94];
- garanta a implementação de subsistemas básicos reusáveis, analisando-os quanto a métricas de reúso como as descritas em [Poulin94] (por exemplo: ser o mais geral possível, não empregar variáveis globais, ser parametrizável, possuir boa documentação);
- controle o conjunto dos subsistemas básicos já existentes e efetive sua aplicação nos novos sistemas. A identificação de subsistemas que mais se adaptem a um ou outro requisito não é uma tarefa fácil. Um suporte automatizado, como um sistema de biblioteca com assitente que apóie a busca de um subsistema com uma dada funcionalidade [Merkl94], pode ajudar;
- gerencie as diversas versões de subsistemas básicos e os sistemas nos quais estão empregadas. Uma correção em um desses subsistemas implica sinalizar esse fato às equipes responsáveis pela manutenção dos outros sistemas que o utilizam.

No entanto, poucas organizações estão preparadas para suportar o ônus na implementação e validação de componentes reusáveis. Adicionar generalidade aos subsistemas e documentá-los dentro de um formato padrão para pesquisa futura é raramente feito devido a pressões de prazo [Panel94].

Considerando que cada subsistema básico é primordialmente derivado a partir de um Tipo de Entidade e de um conjunto de Atividades Essenciais, o reúso de um subsistema básico pode implicar o reúso de toda a modelagem dos eventos externos tratados pelas Atividades Essenciais encapsuladas no subsistema. O modelo associado a esses eventos externos corresponde a um segmento bem definido da Modelo da Essência, da especificação detalhada e do design do subsistema básico. Nessas condições, observa-se uma contribuição para a reengenharia do sistema. Num processo de reengenharia visando à recuperação da essência, ao serem identificados um ou mais eventos externos e o subsistema básico que os implementa, é possível a identificação de outros eventos externos, também tratados pelo mesmo subsistema básico, relevantes para o Modelo da Essência mas ainda não identificados. É claro que o reúso pode ser pensado em nível de abstração ainda mais elevado e aplicar-se à própria modelagem do contexto do Modelo da Essência. Em particular, para problemas análogos, subconjuntos da Lista de Eventos Externos (e toda a cadeia de reações subseqüentes) podem ser reusados na modelagem do novo sistema.

A definição de um método de busca para reúso de toda a modelagem baseada em eventos externos exigirá uma técnica que relacione de forma eficiente:

- subconjuntos de eventos externos;
- sua porções no Modelo da Essência associadas a esses subconjuntos ou deles derivadas, incluindo requisitos funcionais (ativos e passivos) representados no Modelo do Comportamento;
- subsistemas básicos que os implementam.

11.2. Prevenção de Deadlocks

Um ponto relevante durante o design de sistemas concorrentes não é abordado pelo método - **prevenção de Deadlocks.**

O particionamento em subsistemas independentes, com serviços independentes, **evita** a alocação de recursos compartilhados por subsistemas básicos. Mas poderá ocorrer compartilhamento de recursos por Processos de um mesmo subsistema básico e deve ser abordado segundo técnicas tradicionais de prevenção de "deadlocks" [Tanenbaum87]. Ao tratar os subsistemas básicos independentemente, reduz-se o universo de trabalho a sistemas menores (cada subsistema básico) o que facilita a trabalho do projetista em prever pontos de design passíveis de "deadlock".

Entretanto, entre subsistemas básicos podem ocorrer situações do seguinte tipo:

- O subsistema A possui 2 Processos A1 e A2 que compartilham recursos;
- A1 pede um Serviço com resposta associada a um Processo B1 do subsistema B;
- B1 pede um Serviço com resposta associada a um Processo C1 do subsistema C;
- C1 pede um serviço com resposta associada a A2 que necessita de recurso alocado a A1;
- os processos A1, A2, B1 e C1 assim entrarão em deadlock.

É fundamental evitar requisições de serviços encadeados que retornem a um dos subsistemas já participantes.

O método apresentado neste trabalho não fornece qualquer auxílio efetivo para identificação de situações como a descrita acima. Dependerá do emprego de técnicas formais, tais como as que empregam Redes de Petri [Ayache82], que permitam simular o comportamento do sistema visando à detecção de fontes de "deadlocks". No entanto, conforme já foi mencionado, a característica do particionamento e abstração empregados, resultam em uma estrutura de sistema que restringe a verificação da existência de possíveis pontos de dealocks a universos mais simples, isto é, verificações entre requisições de serviços por subsistemas básicos (como no exemplo acima) e verificações de compartilhamento de recursos por processos contidos em subsistemas básicos.

12. Conclusão

A técnica de design proposta neste trabalho permite a transição suave do Modelo da Essência para um design do sistema que garanta o encapsulamento de dados e a identificação de classes que permitam uma implementação orientada a objetos. A implementação dessas classes através de subsistemas básicos permite em muitos casos melhor desempenho sem prejuízo do reúso desses subsistemas e sem incorporar seqüencializações não naturais à aplicação.

A implementação de sistemas em termos de um conjunto de subsistemas básicos independentes conduz, com o passar do tempo e com uma boa organização do ambiente de trabalho, dispor de

uma biblioteca significativa de subsistemas básicos. Esses subsistemas constituem unidades reusáveis de médio porte [Panel94] cujo emprego permite ganhos de produtividade através do reúso de software. A existência de uma biblioteca desse tipo favorece a mudança de cultura no processo de desenvolvimento de sistemas em direção à prática do reúso. A construção do projeto básico, visando ao reúso de subsistemas básicos existentes, permite agilizar o processo de desenvolvimento e ganhar em eficiência pois erros internos a subsistemas amplamente reusados - e, portanto, testados - ocorreriam em número cada vez menor.

As ferramentas de design definem também dois enfoques distintos sobre o particionamento do sistema. Um define sua estrutura e o outro a concorrência existente entre os componentes dessa estrutura. A definição de estrutura emprega Diagramas de Estrutura (DE) que apresentam a versão estática do relacionamento entre subsistemas. A segmentação em subsistemas segue os critérios de encapsulamento de informação, sincronismo através de troca de mensagens e reúso. A especificação da concorrência emprega Diagramas de Comportamento (DC) capazes de mostrar o paralelismo entre serviços de subsistemas a cada nível da hierarquia de decomposição. A relação entre Atividades Essenciais fica explicitada nos DC's permitindo avaliar de temporização dessas atividades para atenderem aos requisitos de desempenho do sistema.

A rápida obsolescência do hardware e a necessidade de atingir rígidos requisitos de desempenho implicam a obrigatoriedade da portatilidade do software de aplicação em relação a diversas arquiteturas de hardware. A mudança de arquitetura de hardware, dentro da técnica aqui apresentada, resulta apenas na revisão do software da interface de compatibilização, correspondendo a um custo desprezível em relação ao tempo gasto no desenvolvimento da camada de aplicação para sistemas de grande porte e complexidade.

A documentação que representa o Modelo da Implementação do sistema pode ser gerada simultaneamente ao desenvolvimento e as características de segmentação da técnica de design podem ser refletidas na documentação do design do sistema. É possível dividi-la em dois níveis de hierarquia, permitindo fácil acesso ao pontos específicos do modelo. As características do método e da documentação resultante possibilitam o uso de técnicas gerenciais e o controle de mudanças nas especificações e interfaces dos subsistemas básicos. Facilita-se o trabalho baseado em equipes independentes pois cada subsistema básico possui uma especificação que delimita sua fronteira, define sua interface e determina a opção de implementação a ser empregada. Qualquer mudança na interface só pode ser efetuada de posse do Projeto Básico, que fornece uma visão global da integração do sistema a partir de subsistemas básicos. Dessa forma, é possível avaliar as conseqüências de alterações introduzidas em subsistemas e, a nível gerencial, determinar providências necessárias; mudanças em interfaces de subsistemas básicos só poderão ser autorizadas pela gerência geral do projeto. Certamente, o uso de hipertexto facilitaria a edição e consulta do tipo de documentação proposto. No entanto, o controle gerencial de mudanças é fundamental para a manutenção da consistência entre a documentação e a implementação resultante para cada subsistema básico.

A Especificação Detalhada dos Subsistemas Básicos, como documento independente produzido em uma segunda etapa do design, favorece o adiamento da especificação de algumas características de implementação do sistema. Isso beneficia o desenvolvimento pois raramente são conhecidos todos os detalhes no início de um projeto, principalmente quanto a interligação com sensores. Dessa forma, uma estratégia de Desenvolvimento Incremental pode ser aplicada, resultando em um sistema que melhor se adapte às necessidades do cliente/usuário.

A técnica de design aqui proposta favorece a concepção de subsistemas básicos de maior nível de abstração, isto é, que reflitam basicamente a implementação dos requisitos registrados na

modelagem conceitual. As dependências em relação à arquitetura de hardware, software e requisitos não essenciais de tolerância a falhas e desempenho limitam-se à configuração do sistema e à Interface de Compatibilização. Isso permite o reúso dos subsistemas a partir de uma análise de eventos externos tornando desnecessária a revisão de código fonte.

Bibliografia

[Akerman91] Akerman C. L.; Modelagem da Essência de Sistemas de Tempo-Real - Estudo de Caso para um Sistema de Controle e Monitoramento de um Processo de Litografia por Feixe de Elétrons; Trabalho de Disciplina de Mestrado na Área de Engenharia de Software; 1991.

[Ayache82] Ayache J.M., Azéma P., Diaz M.; Towards Fault Tolerant Real Time Systems by Using Petri Nets; Application and Theory on Petri Nets, Informatik-Fachberichte; 1982

[Barbosa92] Barbosa E., Maffeo B.; Modelagem da Essência de um Sistema de Controle para uma Área de Linhas de Engarrafamento; Monografias em Ciência da Computação 29/92, Departamento de Informática, PUC- Rio; 1992.

[Brooks87] Brooks F. P.; No Silver Bullet - Essence and Accidents of Software Engineering; IEEE Computer; abril de 1987.

[Bruyn88] Bruyn W., Jensen R., Keskar D. e Ward P.T.; "ESML: An Extended System Modelling Language Based on Data Flow Diagram"; ACM Sigsoft, Software Engineering Notes, 13(1): 58-62; janeiro de 1988.

[Bustard88] Bustard D., Eldore J. e Welsh J.; Concurrent Program Structure; Prentice Hall; 1988.

[CCCL93] Carneiro L.M.F., Coffin M.H., Cowan D.D., Lucena C.J.P.; User Interface High-Order Architectural Models; Technical Report 93-14, Computer Science Departament, University of Waterloo, Waterloo, Ontario, Canada; 1993.

[Chen76] Chen P.P.S.; The entity-relationship model - toward a unified view of data; ACM Transactions on Database Systems, 1(1): 9-36; 1976.

[CILS93a] Cowan D.D., Ierusalimschy R., Lucena C.J.P., Stepien T.M.; Abstract Data Views; Structured Programming, 14(1):1-13; janeiro de 1993.

[CILS93b] Cowan D.D., Ierusalimschy R., Lucena C.J.P., Stepien T.M.; Abstract Data Views; Constructing Composite Applications from Interactive Components; Software Practice and Experience, 23(3):255-276; março de 1993.

[Clemente92] Clemente K.; Modelagem de Sistemas Sócio-Técnicos: Estudo de Caso de um Piloto Automático para Automóvel; Tese de Mestrado, Departamento de Engenharia Elétrica, PUC-Rio; abril de 1992.

[Cowan95] Cowan D.D., Lucena C.J.P.;Abstract Data Views: An Interafce Specification Concept to Enhance Design for Reuse; IEEE Transactions on Software Engineering, vol21, nº3, 1995.

[ESPRIT89] Software Research in ESPRIT's Second Phase; IEEE Software; novembro de 1989.

[Fairley85] Fairley R.E.; Software Engineering Concepts; MacGraw- Hill, 1985.

[Frakes91] Frakes W.B. ; Software Reuse: Is It Delivering.; Proceedings of 13th Annual

International Conference of Software Engineering; maio de 1991.

[Gilbert93] Gilbert J. W.; A concurrent object model for an industrial process-control application; Journal of Object-Oriented Programming; novembro-dezembro de 1993.

[Gomma93] Gomma H.; Software Design Methods for Concurrent and Real- Time Systems; Addison Wesley; 1993.

[HIL92] Hill R. D.; The Abstraction-Link-View Paradigm: Using Constraints to Connect User Interfaces to Applications; CHI'92: 335-342, ACM; maio de 1992.

[IEEE84] Guide to Software Requirements Especifications; ANSI/IEEE STD 830; Julho 1984.

[Jackson83] Jackson M.; System Development; Prentice Hall; 1983

[KP88] Krasner G.E., Pope S. T.; A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80; JOOP: 26-49; agosto-setembro de 1988.

[Kramer85] Kramer J., Magge J.; Dynamic Configuration for Distributed Systems; IEEE Transactions on Software Engineering, vol. SE-11, nº 4; abril de 1985.

[Kramer92] Kramer J., Magee J., Sloman M.; Constructing a Distributed Systems in Conic; IEEE Transactions on Software Engineering Journal vol 7, 2; março de 1992.

[Leveson90] Leveson, N. G.; The Challenge of Building Process-Control Software; IEEE Software; nov. de 1990.

[Leveson91] Leveson, N. G.; Software Requirements Analysis for Real-Time Process-Control Systems; IEEE Transactions on Software Engineering, vol. 17, nº 3; março de 1991.

[Maffeo90] Maffeo B., Ritto A.C.A.; O Esquema Semântico no Modelo do Contexto de um Sistema Computacional; XXIII Congresso de Informáica da SUCESU; 1990.

[Maffeo91] Maffeo B.; ESML: Uma Revisão de Apresentação, Estrutura, Notação e Conteúdo; Monografias em Ciência da Computação 1/91, Departamento de Informática, PUC-Rio;1991.

[Maffeo92] Maffeo B.; Engenharia de Software e Especificação de Sistemas; Editora CAMPUS, 1992.

[Magee94] Magee J., Dulay N., Kramer j.; A Constructive Development Environment for Parallel and Distributed Programs; Proceedings of Second International Workshop on Configurable Distributed Systems; março de 1994.

[Manual84] Manual de Normas do Centro de Apoio a Programação; Publicação Interna da Diretoria de Armamento e Comunicações da Marinha do Brasil; 1984.

[Mcmenamin84] McMenamin M., Palmer J. F.; Essential Systems Analysis; Yourdon Press; 1984.

[Menascé85] Menascé D., Almeida V. A. S.; Planejamento de Capacidade de um Sistema de Computação - Análise Operacional como Ferramenta; Editora Campus 1º edição; 1985.

[Merkl94] Merkl W.; Learning de Semantic Similarity of Reusable Software Componens; Proceedings of Third International Conference on Software Reuse - Advances in Software Reusability, Rio de Janeiro; novembro de 1994.

[NRL] Naval Research Lab Method; comunicação pessoal de Gomma, por ocasião de Tutorial em Software Design Methods for Reusable Concurrent and Real-Time Systems; Third International Conference on Software Reuse, Rio de Janeiro, Brasil; novembro de 1993.

[Panel94] Griss M.L.; KozaczynskiW. V.; Wasserman A. I.; Jette Chirstina; Troy Robert; Panel: Object-Oriented Reuse; Proceedings of Third International Conference on Software Reuse - Advances in Software Reusability, Rio de Janeiro; novembro de 1994.

[Parnas] Parnas D.;Information Distribution Aspects of Design Methodology; IFIP Congress Proceedings; Ljubljana, Yugoslavia; 1971.

[Perrow84] Perrow C.; Normal Accidents; Basic Book, New York; 1984.

[Poulin94] Poulin J.S.; Measuring Software Reusability; Proceedings of Third International Conference on Software Reuse - Advances in Software Reusability; novembro de 1994.

[Richter92] Richter G., Maffeo B.; Towards a rigorous interpretation of ESML - Extended System Modeling Language; IEEE Transactions on Software Engineering, vol. 19 nº 2 pg165; 1993.

[Ritto91] Ritto A.C.A, Maffeo B. ; Definindo o Problema a ser Tratado por um Sistema Computacional - Modelo do Contexto; Monografias em Ciência da Computação 1/91, Departamento de Informática, PUC-Rio; 1991.

[Rumbaugh91] Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W.; Object-Oriented Modeling and Design; Prentice-Hall; 1991.

[Sanchez93.1] Sanchez M. L., Maffeo B.; Gerência de Projeto Orientado a Encapsulamento de Dados e a Troca de Mensagens entre Subsistemas Autônomos; Anais do VII Simpósio Brasileiro de Engenharia de Software; outubro de 1993.

[Sanchez93.2] Sanchez M. L., Maffeo B. ; Ferramentas e Técnicas para a Modelagem da Essência de Sistemas de Tempo-Real para Controle e Monitoramento de Processos; Série Monografias em Ciência da Computação 14/93, Departamento de Informática, PUC-Rio; 1993.

[Sanchez94] Sanchez M. L., Maffeo B., Leite J. C. S. P. ; Ferramentas e Técnicas para a Modelagem da Essência de Sistemas de Tempo-Real para Controle e Monitoramento de Processos; Anais do 10º Congresso Brasileiro de Automática e do 6º Congresso Latino Americano de Controle Automático, Rio de Janeiro; setembro de 1994.

[Tanenbaum87] Tanenbaum A.S.; Operating Systems: design and Implementation; Englewood Cliff, N.J., Prentice Hall; 1987.

[Tracz94] Tracz W.; Reuse State of the Art and State of the Practice Report Card; Proceedings of Third International Conference on Software Reuse - Advances in Software Reusability, Rio de Janeiro; novembro de 1994.

[Ward84] Ward P. T.; System Development Without Pain; Yourdon Press;1984.

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

[Wentzel94] Wentzel K.; Software Reuse, Facts and Myths; Proceedings of 16th International Conference on Software Engineering; maio de 1994.

[Yourdon89] Yourdon E.; Modern Structured Analysis; Yourdon Press;1989.