MAURO NOTARNICOLA MADEIRA

ORIENTAÇÃO POR OBJETOS APLICADA À AUTOMAÇÃO INDUSTRIAL : OBJETO-IMAGEM

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Curso de Pós-Graduação em Ciências da Computação, Universidade Federal de Santa Catarina.
Orientador : Prof. Rogério Cid Bastos.

FLORIANÓPOLIS

1995

ORIENTAÇÃO POR OBJETOS APLICADA À AUTOMAÇÃO INDUSTRIAL: OBJETO-IMAGEM

MAURO NOTARNICOLA MADEIRA

ESTA DISSERTAÇÃO FOI JULGADA ADEQUADA PARA A OBTENÇÃO DO TÍTULO DE

MESTRE EM CIÊNCIAS DA COMPUTAÇÃO

E APROVADA EM SUA FORMA FINAL PELO PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Prof. Rogério Cid Bastos, Dr. Eng.

Coordenador do Curso

Banca Examinadora:

Prof. Rogério Cid Bastos, Dr. Eng.

Orientador (Presidente)

Prof. Arno Bollmann, Dr.-Ing.

Prof. Fernando Alvaro Ostuni Gauthier, Dr. Eng.

Prof. Sérgio Peters, Dr. Eng.

Florianópolis, 28 de agosto de 1995

À minha mãe

Shirlei Notarnicola Madeira

Ao meu amigo

Jorge Fernando Silva de Araujo

À memória de meu orientador

professor Hermann Adolf Harry Lücke

AGRADECIMENTOS

O presente trabalho não teria sido levado a termo satisfatório sem o apoio inestimável de pessoas e instituições. Entre as pessoas cumpre agradecer

Ao professor **Hermann Adolf Harry Lücke**, pelos incontáveis momentos de orientação, atenção e amizade, que mesmo nas etapas mais difíceis, trabalhou sempre de forma otimista.

Ao professor **Victor Juliano De Negri**, pelas discussões construtivas durante a elaboração deste trabalho.

À **Andrea Luswarghi de Souza**, pelo encorajamento e confiança de que tudo daria certo.

À professora **Clara Amélia de Oliveira** pelo apoio espiritual e material concedido em um dos momentos mais difíceis, permitindo a continuação do trabalho.

Ao professor **Walzi Conceição Sampaio da Silva** pelo entusiasmo e apoio.

Ao professor **Renato M. Doria** pelo acreditar na ciência e me dar inspiração.

Ao professor **Rogério Cid Bastos** pelas contribuições na etapa final do trabalho.

À **Vera Lúcia** pela sua ajuda em vários momentos.

Ao amigo **José Antônio Alves Duarte** pelo suporte técnico.

No que concerne às instituições agradeço ao programa de mestrado da Universidade Federal de Santa Catarina pelo ambiente estimulante que foi fundamental para o desenvolvimento de minha pesquisa. Agradeço também à Capes que subvencionou a etapa que ora se encerra.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

A/D	-ANALÓGICO/DIGITAL
ASCII	**-AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE** - CÓDIGO PADRÃO PARA TROCA DE INFORMAÇÃO AMERICANO
BCD	**-BINAY CODED DECIMAL** - DECIMAL CODIFICADO EM BINÁRIO
BIT	**-BINARY DIGIT** - DÍGITO BINÁRIO
BYTE	-CONJUNTO DE 8 BITS
CA	-CORRENTE ALTERNADA
CC	-CORRENTE CONTÍNUA
D/A	-DIGITAL/ANALÓGICO
DFD	**-DATA FLOW DIAGRAM** - DIAGRAMA DE FLUXO DE DADOS
DMA	**-DIRECT MEMORY ACCESS** - ACESSO DIRETO À MEMÓRIA
DSP	**-DIGITAL SIGNAL PROCESSING** - PROCESSAMENTO DIGITAL DE SINAL
E/S	-ENTRADA/SAÍDA
EBCDIC	**-EXTENDED BINARY CODED DECIMAL INTERCHANGE CODE** - CÓDIGO DE TROCA BINÁRIO PARA DECIMAL ESTENDIDO
ER	**-ENTITY RELATIONSHIP** - RELACIONAMENTO ENTRE ENTIDADES
FT	-FUNÇÃO DE TRANSFERÊNCIA
IFD	**-INFORMATION FLOW DIAGRAM** - DIAGRAMA DE FLUXO DE INFORMAÇÕES
LED	**-LIGHT EMITTER DIODE** - DIODO EMISSOR DE LUZ
LSB	**-LEAST SIGNIFICANT BIT** - BIT MENOS SIGNIFICATIVO
NA	-NORMALMENTE ABERTO
O(I)BE(F)	**-OUTPUT(INPUT) BUFFER EMPTY (FULL)** - BUFFER DE SAÍDA (ENTRADA) VAZIO (CHEIO)
OFD	**-OBJECT FLOW DIAGRAM** - DIAGRAMA DE FLUXO ORIENTADO POR OBJETOS
OMT	**-OBJECT MODELIG TECHNIC** - TÉCNICA DE MODELAGEM POR OBJETOS
OOD	**-OBJECT-ORIENTED DESIGN** - PROJETO ORIENTADO POR OBJETOS
PGA	**-PROGRAMABLE GAIN AMPLIFIER** - AMPLIFICADOR DE GANHO PROGRAMÁVEL
S&H	**-SAMPLE AND HOLD**
SA	-SISTEMA DE ATUAÇÃO
SM	-SISTEMA DE MEDIÇÃO
UCP	-UNIDADE CENTRAL DE PROCESSAMENTO

RESUMO

Este trabalho tem por objetivo descrever a criação de modelos de sistemas de automação utilizando o **objeto-imagem** como técnica de abstração. Com este propósito são descritos os principais elementos e técnicas da automação industrial na construção de soluções computacionais e a utilização da orientação por objetos. São apresentadas características dos sistemas automáticos, classificação e comportamento de instrumentos típicos de sistemas de medição e atuação, rotinas e dispositivos de interfaceamento, elementos da orientação por objetos e características da codificação em uma linguagem orientada por objetos. Como resultado tem-se a caracterização e comportamento do objeto-imagem.

ABSTRACT

This work has the purpose to present the production of automatic systems models applying the **image-object** as an abstraction mechanism. To reach this objective the main components and procedures are described to build software solutions and the object-oriented use. Characteristics of automatic systems, classification and behaviour of typical actuator and measure systems instruments, interface concepts, and object-oriented design and coding are shown. As a result the **image-object** artifice and application examples are presented.

1 - Introdução

1.1 - Orientação por objetos

Nos decorrer dos últimos anos o paradigma da orientação por objetos é cada vez mais utilizado na área de desenvolvimento de software. Para contornar a chamada **crise do software** (BOOCH, 1991, JACOBSEN et al., 1992) utiliza-se as características da orientação por objetos nas fases de especificação, projeto, implementação, testes e manutenção, permitindo:

- Especificação mais fácil através da ênfase na descrição conceitual do problema e relegando para um segundo plano a implementação;
- O desenvolvimento incremental;
- A diminuição da complexidade e interdependência alcançada pela utilização da classe/objeto, que encapsula dados e operações;
- A manutenção mais fácil obtida com o uso do polimorfismo;
- A construção de componentes reutilizáveis através do mecanismo de herança entre classes.

Encontra-se razoável bibliografia ilustrando aplicações em sistemas de automação comercial (RUMBAUGH et al., EMBLEY et al., COAD & YOURDON, etc.) , porém poucas na área de automação industrial. Este fato deve-se em parte pelo pouco ou nenhum suporte oferecido pelas linguagens puramente orientadas por objetos para programação em tempo real e manipulação direta de elementos de hardware, como no interfaceamento de processos.

1.2 - Motivações e objetivos do trabalho

Este trabalho foi motivado pelos seguintes fatores:

- A orientação por objetos tem sido uma tendência no desenvolvimento de software, no entanto existem poucas informações sobre aplicações voltadas para a automação de sistemas técnicos.

- Os poucos exemplos encontrados como nos livros de BOOCH (1991), COAD & YOURDON (1991), RUMBAUGH et. al. (1991), EMBLEY et al. (1992), FIEDLER et. al. (1991), JACOBSEN et al. (1992), fornecem apenas aspectos e exemplos parciais.

- É possível verificar ainda, que os sistemas de medição e atuação e seus componentes não têm o mesmo tratamento dispensado aos outros elementos de um sistema automático.

Frente a esta situação emergem os objetivos principais do trabalho:

- a) Abstrair para o meio computacional elementos que compõem os sistemas de medição e atuação e suas relações, empregando o paradigma da orientação por objetos e a conceituação do **objeto-imagem**;

- b) Exemplificar um projeto, representativo para a área de automação industrial, direcionado para sistemas hidráulicos.

1.4 - Apresentação do trabalho

Para atingir os objetivos descritos, foram realizados vários trabalhos que são apresentados em 5 itens:

1. Introdução sobre as características de sistemas técnicos automáticos, definição de sistemas de medição e atuação e ponto de partida para seu detalhamento;

2. Descrição dos componentes da parte física dos sistemas de aquisição e atuação, descrição dos elementos e comportamento do interfaceamento de processo;

3. Descrição dos dispositivos de interfaceamento com o computador, incluindo formas de sincronismo, características dos dados e métodos de transferência;

4. Apresentação dos principais aspectos do paradigma da orientação por objetos e sua inclusão no ciclo de vida do software;

5. Características da implementação orientada por objetos, incluindo o uso da linguagem C++ e testes de desempenho;

Estes cinco itens formam a base para a conceituação do **objeto-imagem** e o projeto e implementação de sistemas de medição e atuação. Ao final do trabalho é apresentada a utilização destes conceitos em um exemplo em fase de desenvolvimento.

2 - Sistemas Automáticos

2.1 - Introdução

Este capítulo utiliza como ponto de partida o trabalho de LÜCKE & DE NEGRI (1994), onde desenvolvem-se a formalização do projeto de sistemas automáticos e uma metodologia para a construção de modelos que incorporam os requisitos de projeto e servem de base para descrições mais detalhadas. O modelo proposto pelos autores e sua forma de representação proporcionam a identificação dos subsistemas e recursos presentes em um sistema automático, utilizado neste trabalho como base para a conceituação do **objeto-imagem** e composição da parte de **software** dos sistemas de medição e atuação.

2.2 - Sistemas de automação industriais

Segundo FREDERICK & CARLSON (1971) um sistema é a coleção de objetos interagentes, designado para alcançar um objetivo específico ou um conjunto de objetivos para processar a manipulação e controle de recursos como **matéria, energia e informação**.

A figura 2.1 esquematiza um sistema técnico utilizando-se da notação em rede de Petri canal/agência (HANISH, 1992, p. 81) (HEUSER, 1991), onde recursos de matéria, energia e informação são representados por círculos e processos que consomem ou produzem recursos por retângulos.

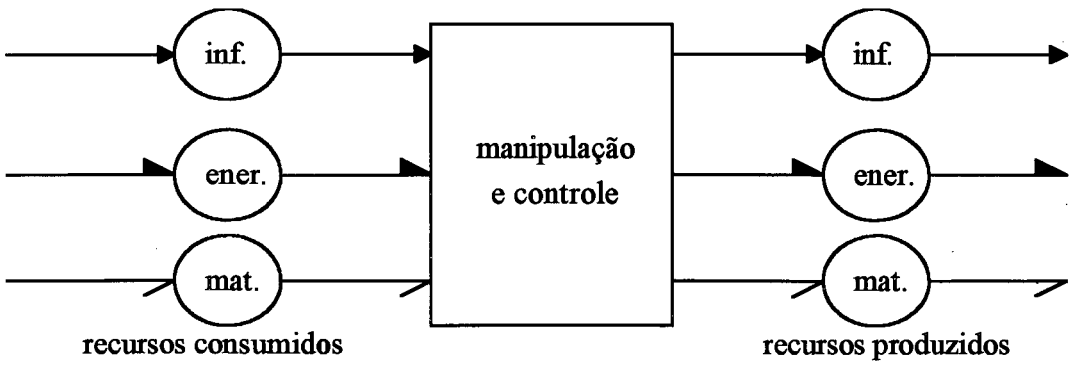


Figura 2.1 - Modelo de um sistema técnico (LÜCKE & DE NEGRI, 1994)

Partindo-se deste modelo, identifica-se quais recursos devem efetivamente ser processados para atender ao objetivos de um sistema automático.

No trabalho de LÜCKE & DE NEGRI, um sistema automático pode ser então dividido em duas partes, onde uma utiliza como recurso **energia/matéria** e a outra a **informação**. As duas partes obtêm estes recursos do meio ambiente e interagem através de informações (fig.2.2) (LÜCKE e DE NEGRI, 1994).

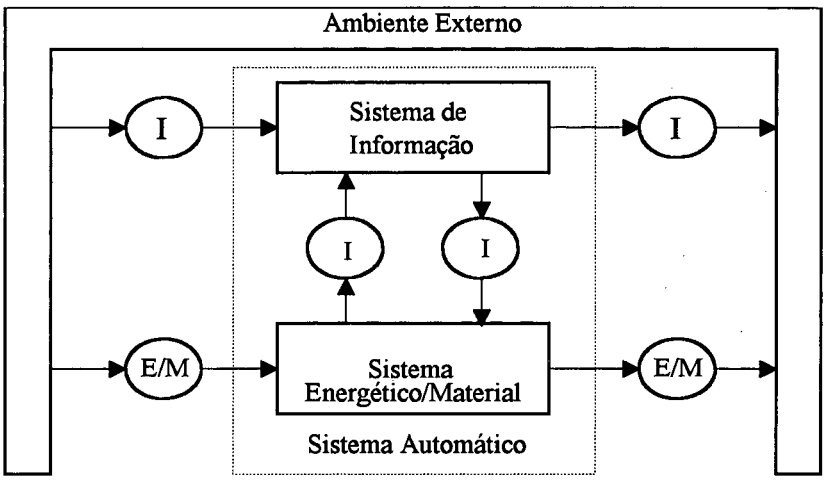


Figura 2.2 - Sistema Automático (LÜCKE & DE NEGRI, 1994)

O sistema Energético/Material é composto de elementos físicos que processam recursos energéticos e materiais, realizando uma tarefa determinada. A automatização é feita sobre a observação e controle de grandezas específicas deste processo, ou seja, seus fenômenos físico/químicos (rotação, vazão, temperatura, Ph, etc.). O sistema de informação é composto por um computador e um programa coordenador de atividades, extraindo informações do sistema energético/material, processando-as e introduzindo modificações em resposta.

Para que o sistema de informação possa interagir com o sistema energético/material é necessário o estabelecimento de canais de comunicação entre eles. Na figura 2.3 apresenta-se o detalhamento do modelo apresentado, onde os canais de informação são apresentados como subsistemas de medição (SM) e subsistemas de atuação (SA), que possuem partes no sistema de informação e partes no sistema energético/material.

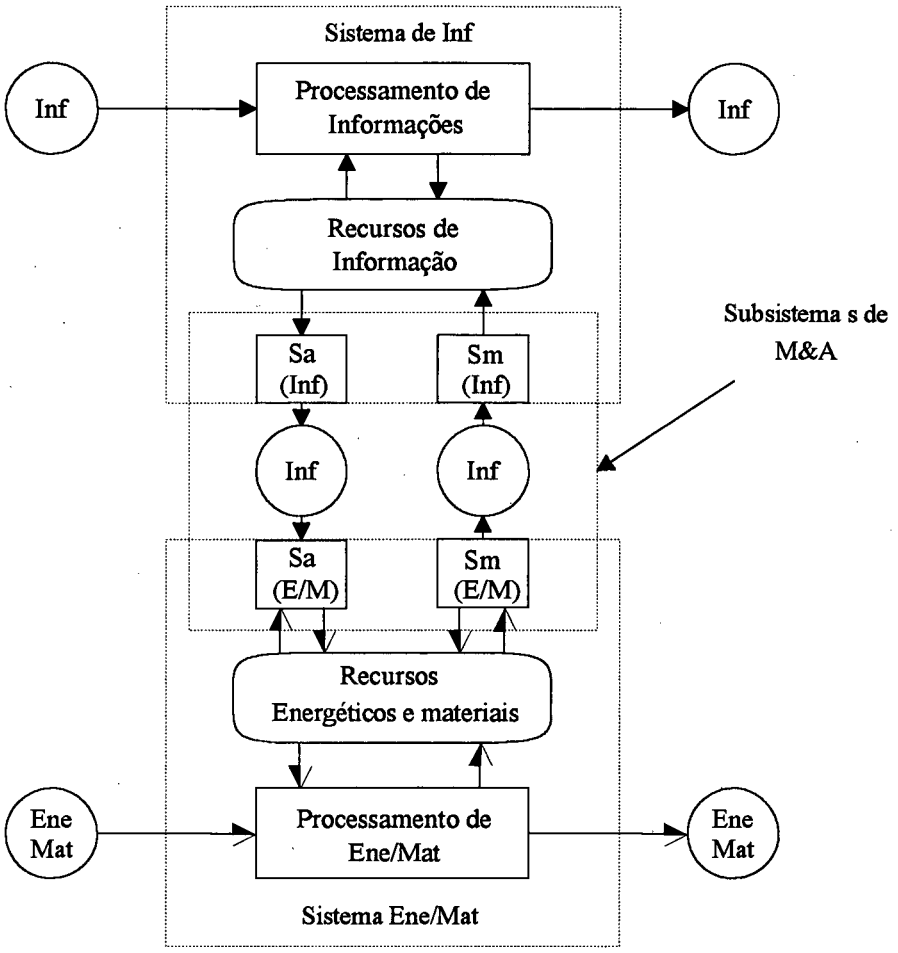


Figura 2.3 - Modelo canal/agência detalhado

Com base nestes novos elementos e centrando a atenção nos subsistemas de medição e atuação, um sistema automático pode ser dividido em quatro partes:

- O meio físico, ou sistema energético material, do qual o sistema extrai informações e atua;
- O subsistema de medição (SM), que capta informações sobre grandezas do meio físico e as envia para o computador;
- O subsistema de atuação (SA) que permite ao computador introduzir alterações em grandezas do meio físico;
- O computador, ou sistema de informação, que interage entre a medição e a atuação, gerenciando o funcionamento destes.

Na figura 2.4 é representado um modelo de um sistema de medição e atuação em que o computador se caracteriza como o sistema de informação, exercendo o papel central no gerenciamento das informações.

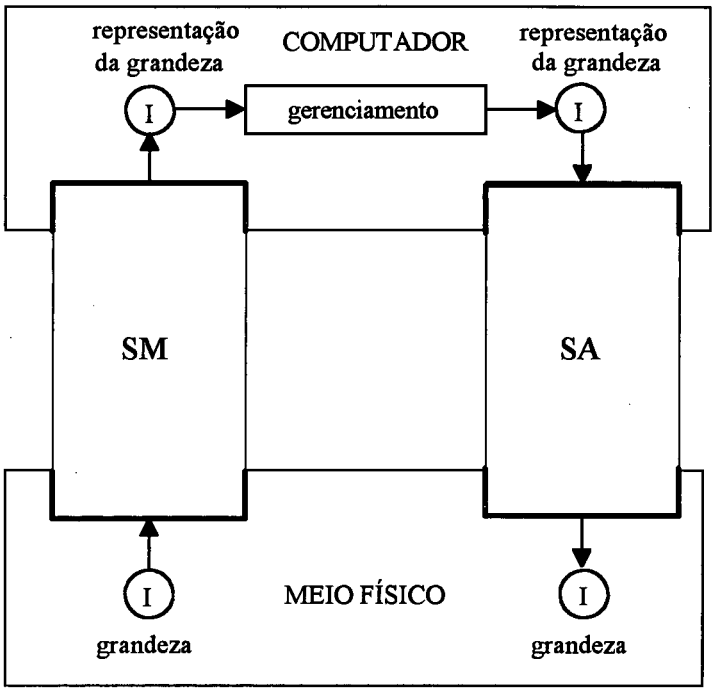


Figura 2.4 - Diagrama de Blocos

Os diversos elementos interagem através de sinais que são os transportadores de informação. Tanto o subsistema de medição quanto o subsistema de atuação são compostos de instrumentos que compatibilizam as diferentes formas que os sinais podem apresentar na transferência de informações entre computador e meio físico. Estas formas são caracterizadas em função da natureza dos fenômenos monitorados/controlados e também dos instrumentos de medição e atuação utilizados. Como pode ser observado na figura 2.4, parte dos sistemas de medição e atuação invadem os limites do computador, dificultando traçar uma fronteira bem definida entre estes, como por exemplo acontece com o emprego de placas de aquisição encaixáveis no sistema eletrônico do computador. O mesmo ocorre com relação ao sistema físico, onde os sensores e atuadores estão imersos. A figura 2.5 ilustra o modelo de um sistema mais detalhado com componentes básicos necessários para uma aplicação em um sistema técnico.

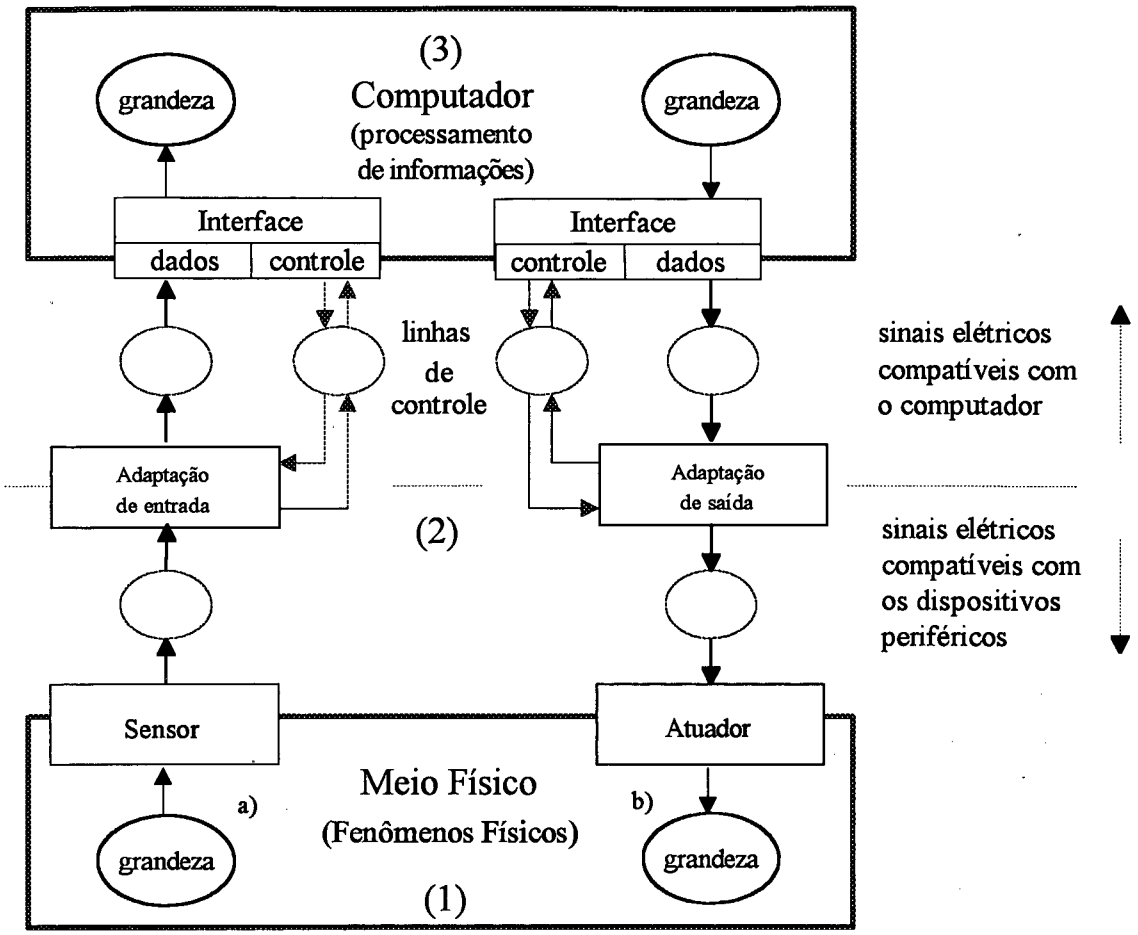


Figura 2.5 - Elementos de um sistema de medição e atuação

Através desta figura pode-se delimitar três níveis de sinais, de acordo com a natureza dos dispositivos e seus sinais característicos:

1. Meio físico -

   a) transformação de uma grandeza da medição de um fenômeno físico em sinais elétricos fornecido pelos sensores;

   b) sinais elétricos sendo transformados em atuação sobre uma grandeza no meio físico através de atuadores.

2. Adaptação -

   Os sinais que provêm dos sensores ou são fornecidos aos atuadores necessitam de adaptações em seus níveis elétricos, sincronismo, freqüência e/ou isolamento, permitindo a manipulação pelo computador.

3. Computador -

   Neste nível, além do processamento da informação sendo recebida e enviada, são disponíveis meios de atuação sobre o próprio processo de adaptação através de canais de controle e verificação do estado dos elementos da adaptação. O componente que fornece esta funcionalidade ao computador é denominado **interface**.

O capítulo 3 detalha os elementos, recursos e técnicas de utilização dos elementos presentes no meio físico e adaptação (níveis 2 e 1). O capítulo 4 apresenta características e comportamentos do interfaceamento e transferêcia da informação digital para o computador.

Os capítulos 5, 6, 7 e 8 propõem soluções para a realização no meio computacional, apresentando o paradigma da orientação por objetos, sua implementação, a conceituação do **objeto-imagem** e sua aplicação.

3 - Sistemas de Medição e Atuação

3.1 - Introdução

Este capítulo descreve os elementos presentes na interface do processo, detalhando os sinais e elementos responsáveis ao seu processamento, partindo do meio físico até a fronteira do computador, no caso da medição, e em sentido contrário para a atuação.

3.2 - Tipos de Sinais

Dependendo da natureza do fenômeno físico e do instrumento responsável pela sua detecção/atuação são encontrados diferentes tipos de sinais. Uma característica predominante está relacionada com a variação da amplitude do sinal em função do tempo, sendo uma das formas em que a informação pode se manifestar (STEIGLITZ, 1974, p.24). Na figura 3.1 são ilustrados exemplos de alguns dispositivos de acordo com o tipo de sinal característico (WECK, 1989, p.85).

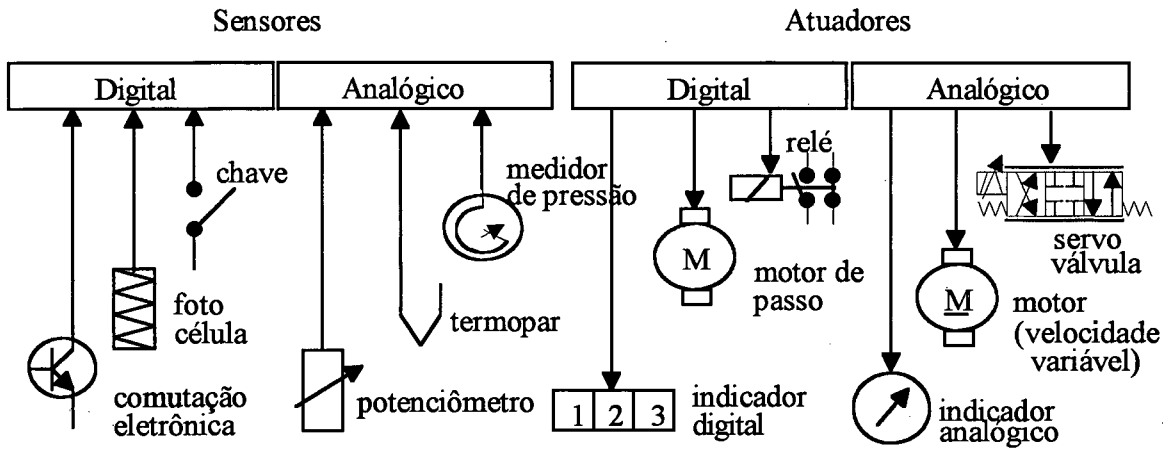


Figura 3.1 - Caracterização do sinal em relação a alguns dispositivos periféricos (WECK, 1989)

Pode-se dividir os dispositivos em dois tipos :

Analógicos - Operam com sinais que transportam informação na variação contínua de sua amplitude. Nos instrumentos de aquisição e atuação, depois de tratados, estes sinais apresentam valores situados geralmente na faixa de -10 a +10 volts ou de 4 a 20 mA .

Digitais - Apresentam sinais discretos ou valores binários de amplitude, que são representados pelos níveis **baixo** e **alto**. Ex.: Após o tratamento, os instrumentos que empregam lógica TTL (Transistor-Transistor-Logic) representam os dois níveis através das tensões:

- 0 V a 0,8 V para o nível baixo
- 3.0 V a 5.0 V para o nível alto

Sinais digitais também são denominados sinais discretos. Na utilização prática como entrada, os sinais digitais podem representar o estado de um ou mais chaves elétricas, a existência de tensão em determinado ponto de um circuito, um botão pressionado, etc. Saídas digitais podem controlar uma lâmpada, a posição de um motor, o acionamento de uma válvula pneumática direcional, a ligação de uma bomba hidráulica, etc.

Uma forma variante do sinal digital é denominado **pulso**. Neste caso a informação está contida no número de transições de estados que ocorreram ou na taxa em que estas transições estão ocorrendo (pulsos/seg.). Dispositivos de contagem de tempo e freqüencímetros produzem este tipo de informação. Na figura 3.2 são apresentados gráficos com exemplos de um sinal analógico e digital (tensão X tempo). O sinal analógico é contínuo em amplitude e tempo. O digital é discreto na amplitude e dependente do sincronismo para ser contínuo ou discreto no tempo.

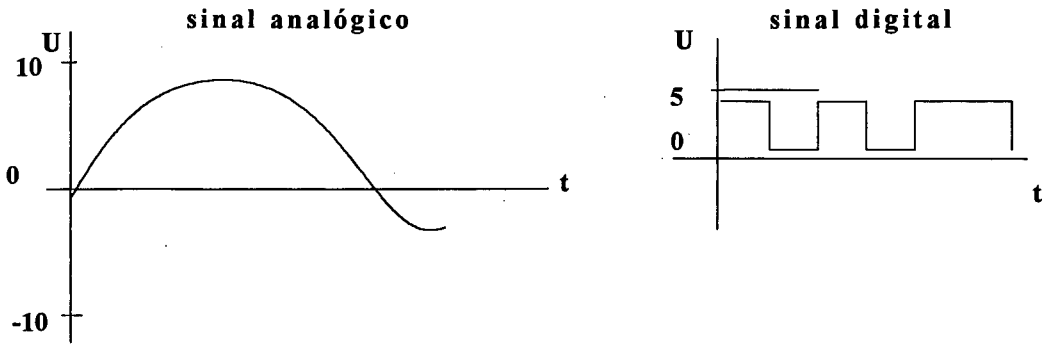


Figura 3.2 - Exemplos dos sinais analógico e digital

3.3 - Sistema de Medição

O conjunto de dispositivos envolvidos na coleta de informação é denominado **sistema de medição (SM)**. Este sistema promove adaptações ao sinal coletado no meio físico, onde as mais simples envolvem a aquisição de sinais de origem digital e as mais complexas os sinais analógicos de alta freqüência. A figura 3.3 ilustra o exemplo de um conjunto de elementos para a aquisição de um sinal digital (caminho à esquerda) e para um sinal analógico (caminho à direita).

O arranjo utilizado para a medição analógica pode variar em função de vários fatores, como velocidade e custo de implementação. Os itens seguintes descrevem os elementos da figura 3.3.

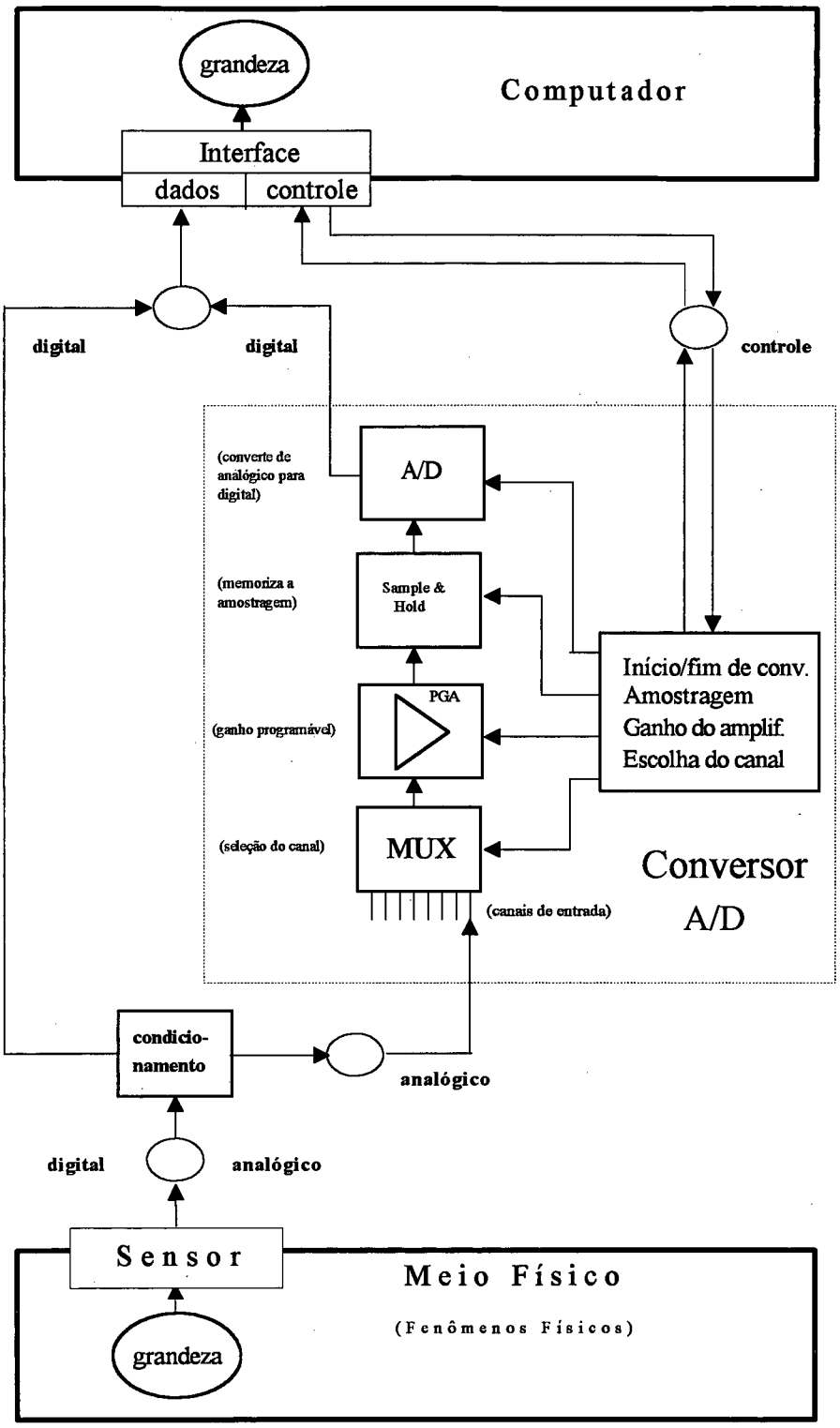


Figura 3.3 - Sistema de Medição

3.3.1 - Sensores

Para que um fenômeno possa ser observado por um sistema computadorizado é necessário sua conversão para um sinal elétrico. O **transdutor** é o dispositivo que converte uma forma de energia em outra, como por exemplo uma pressão na deformação de um diafragma, vazão em rotação de uma hélice, temperatura em uma corrente elétrica, ou qualquer outro tipo de combinação. O conjunto transdutor que apresenta em sua saída um sinal elétrico é denominado **sensor elétrico**, ou simplesmente **sensor** (SMITH, 1984, p.34).

Os sensores são divididos em dois tipos fundamentais (DOEBLIN, 1983, p. 15)[1]:

- **Passivos** - Produzem uma tensão ou corrente de saída em função da grandeza sob medição. A energia de saída é fornecida totalmente, ou quase totalmente, pelo sinal de entrada;
- **Ativos** - Necessitam de uma fonte auxiliar de energia que fornece a maior parte da energia de saída, enquanto o sinal de entrada contribui com uma parte insignificante. Neste caso o fenômeno medido causa a variação de uma grandeza interna do sensor (capacitância, resistência, indutância, etc.) que é convertida em uma tensão ou corrente elétrica de saída.

A tabela 3.1 relaciona alguns sensores classificados de acordo com esta divisão, acrescentando a grandeza que sofre variação.

---

[1] Em algumas obras esta classificação é invertida.

TIPO	ELEMENTO SENSOR	GRANDEZA
Ativo	Piezoelétrico	pressão e força
	Eletrodinâmico	velocidade
	Termopar	temperatura
	Fotoelemento	radiação luminosa
Passivo	Resistivo (variação da resistência por:)	alteração mecânica (geométrica)
		deformação
		campo magnético
		umidade
		temperatura
		luz
	Indutivo (variação da indutância por:)	movimentação relativa do núcleo
		movimentação do objeto sob teste
		alteração da permeabilidade magnética
	Capacitivo (variação da capacitância por:)	variação da distância relativa de suas placas
		área
		dielétrico

Tabela 3.1 - Classificação dos sensores e grandezas sob variação.

3.3.2 - Função de Transferência

A relação entre o fenômeno sob investigação e a saída de um sensor é equacionada através de sua função de transferência nominal (FTn), geralmente linear (SCHNEIDER & LÜCKE, 1989 p. 3.17)

$$L = FTn(GM)$$

onde L é a leitura da medição e GM a grandeza sob medida. Esta equação descreve o **comportamento ideal do sensor**.

A construção prática de um sensor envolve a aplicação de valores conhecidos e anotação das respostas dentro de uma faixa operacional desejada. Este processo visa levantar a função de transferência real (FTR) do sensor, que equivale a observação da função de transferência nominal (FTn) em uma faixa limitada de operação e é denominado **calibração**. A curva real é resultante da interpolação linear ou polinomial das respostas em função da quantidade finita de pontos investigados. Este processo resulta no fornecimento de uma tabela, gráfico ou equação do comportamento do sensor dentro de uma faixa específica (figura 3.4) (SCHNEIDER & LÜCKE, 1989 p. 3.15).

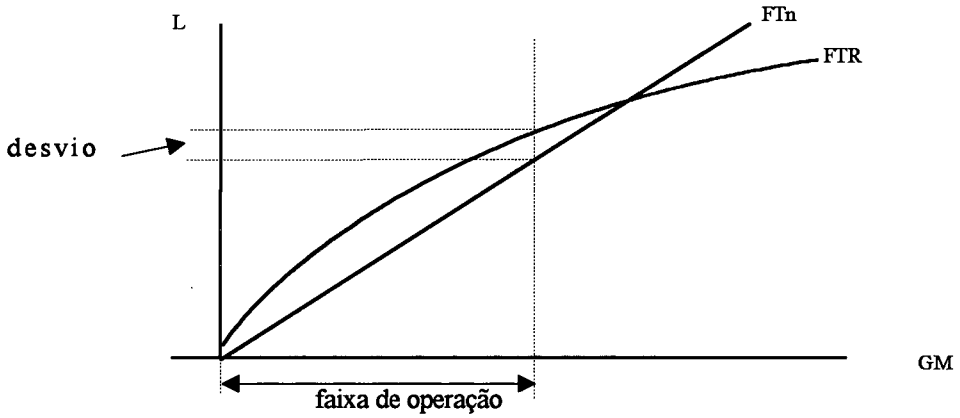


Figura 3.4 - Função de Transferência (SCHNEIDER & LÜCKE, 1989)

3.3.3 - Exemplos de sensores

A seguir são dados como exemplos, o princípio físico, a função de transferência e a faixa de operação de três sensores da tabela 3.1.

3.3.3.1 - Sensor de Temperatura

Os sensores para a medição de temperatura mais empregados são **termopares**, utilizados para medições de temperaturas entre -200°C a +4000°C. Fisicamente são produzidos através da

junção de dois fios de metais diferentes. Esta junção produz uma tensão nos terminais livres dos dois condutores, proporcional a temperatura aplicada à junção (efeito Seebeck). A tensão de saída varia entre -10mV a +50mV e tem uma sensibilidade média de 10 a 50μV/°C, de acordo com os metais empregados. A tensão gerada tem baixa linearidade em função da temperatura aplicada à junção.

Algumas combinações de metais são mais empregadas e possuem letras designativas:

- Tipo J (ferro-constantan[2]) - baixo custo, alta sensibilidade e precisão média. São empregados até 760°C;
- Tipo K (cromo-alumínio) - custo e sensibilidade médios, baixa precisão. Alta faixa de temperatura até 1372°C;
- Tipo T (cobre-constantan) - custo e sensibilidade médios, alta precisão. Utilizado em baixas temperaturas.

A figura 3.5 ilustra o símbolo empregado para a representação de termopares e as curvas características para os três tipos de junções apresentados.

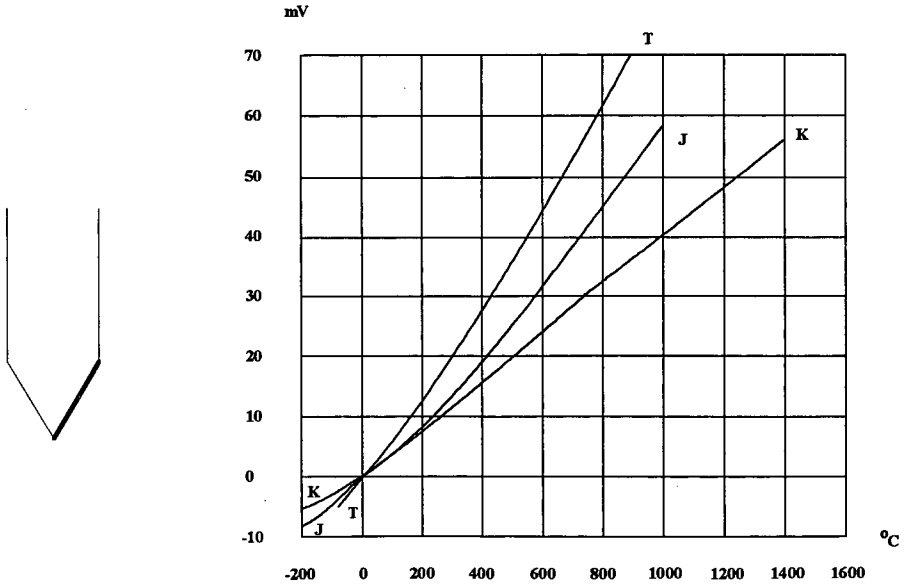


Figura 3.5 - Símbolo e curvas características dos tipo J, K e T

---

2 Constantan - liga composta de cobre e níquel

A função de transferência para uma junção simples pode ser expressa por

$$V = k(t)$$

onde k é o coeficiente de Seebeck definido para o tipo de junção e t é a temperatura em graus Kelvin. Está equação não tem efeito prático, uma vez que novas junções aparecem quando da conexão do termopar ao equipamento de medição. Como o coeficiente k é altamente não linear é comum o emprego de tabelas para a descrição do comportamento de cada tipo de junção. Nas aplicações computacionais os dados extraídos das tabelas geram equações polinomiais entre a quinta a oitava ordem, dependendo da precisão desejada.

Como outras alternativas para a medição de temperatura tem-se:

- Termistor - dispositivo a base óxido de metal ou semicondutor que varia sua resistência elétrica em função da temperatura. Possuem resolução típica de 0,01°C, com precisão 10 vezes maior do que os termopares (±0,1°C). São utilizados para temperaturas de -50 a +100°. Com a possibilidade da combinação de dois dispositivos com coeficientes complementares é possível a construção de dispositivos com alta linearidade.

- Sensor de estado sólido - são dispositivos na forma de circuitos eletrônicos integrados que exploram a influência da temperatura sobre as junções semicondutoras de seus componentes. Possuem alta correção de linearidade e também tensão/corrente de saída já amplificadas. O tipo mais comum tem uma saída de 1μA/Kelvin. A faixa de utilização vai de -50 a +150°C. A precisão alcança ±0,5°C e resolução de até 0,1°C.

3.3.3.2 - Sensor para Deslocamento relativo - longitudinal e angular

A medição do deslocamento é importante por si só como também pelo fato de ser a base de funcionamento de muitos transdutores que efetuam medições de pressão, força, aceleração, temperatura, etc. (DOEBLIN, 1983, p.218). O dispositivo de construção mais simples é o **potenciômetro resistivo**, consistindo basicamente de elemento resistivo provido de um contato móvel (figura 3.6). O movimento do contato pode ser longitudinal, angular ou a combinação de ambos.

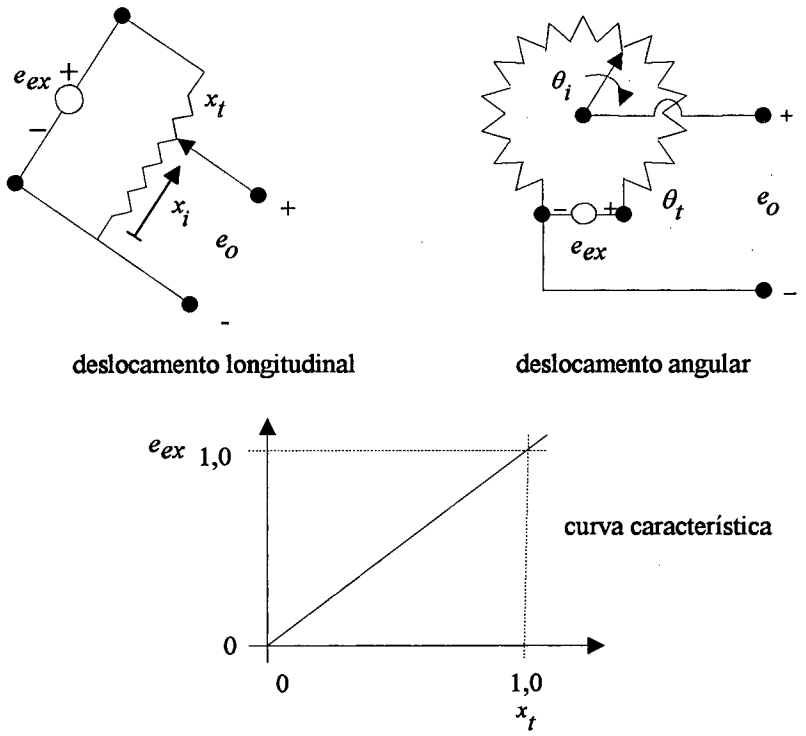


Figura 3.6 - Sensor de deslocamento potenciométrico (DOEBLIN, 1983)

Se a distribuição da resistência é proporcional ao deslocamento, a tensão de saída e_0 irá corresponder ao movimento de entrada x_i ou θ_i

$$e_o = \frac{x_i}{x_t} e_{ex},$$

Onde e_0 é a tensão de saída, x_t a resistência total do potenciômetro e e_{ex} a tensão de alimentação. Este comportamento representa o funcionamento ideal (FTn), neste caso sem carga.

A adição de uma carga de saída faz com que a resposta perca sua linearidade, sendo portanto necessário adicionar uma correção.

Faixas de operação - Na prática estes potenciômetros são construídos para a medição de deslocamentos longitudinais com cursos de 0,002 até 1 m. Para o deslocamento angular são construídos potenciômetros com cursos de 10° até 60 voltas completas.

3.3.3.3 - Sensor para Detecção de Proximidade

A figura 3.7 ilustra um **sensor de proximidade** eletrônico e seu gráfico de operação para o tipo NA (Normalmente Aberto) (OMRON). O alvo é normalmente uma peça metálica ou um veículo metálico que contém a peça que se deseja detectar a posição.

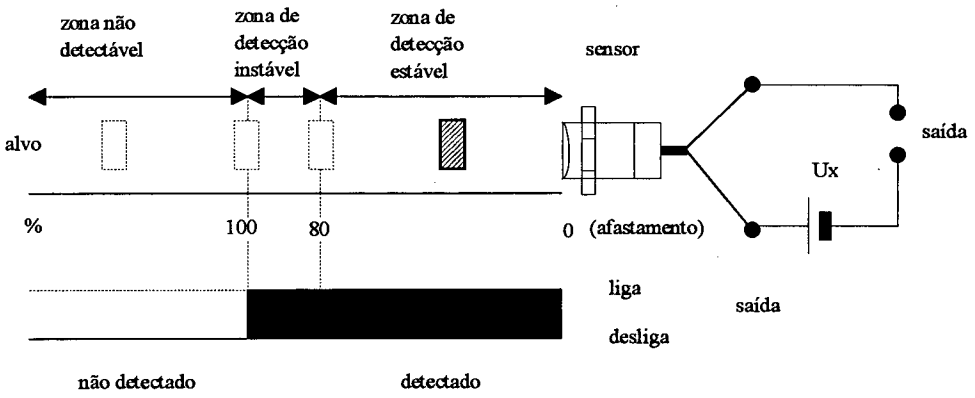


Figura 3.7 - Sensor de proximidade tipo NA (OMRON)

A distância de detecção varia de 2mm a 20mm de acordo com o tipo de sensor. Como o princípio de funcionamento do sensor é digital, é comum a apresentação de uma tabela relacionando sua saída em função da proximidade do alvo sob detecção (tabela 3.2).

Alvo	Saída (volts)
afastado	0.00
detectado	Ux

Tabela 3.2 - Tabela de resposta para o sensor de proximidade do tipo NA.

3.3.4 - Condicionamento do sinal

Após o sensor geralmente o sinal não está em condições de ser utilizado diretamente, passando pela **etapa de condicionamento** (fig. 3.3). Normalmente o condicionamento pode ser dividido em:

- **passivo** - que inclui divisão de voltagens, supressão de transientes, conversão corrente-voltagem e filtragem;
- **ativo** - que inclui amplificação e isolamento.

Para a obtenção de uma melhor relação sinal/ruído, o condicionamento se posiciona o mais próximo ao sensor possível.

3.3.3.1 - Condicionamento de sinais analógicos

A amplitude máxima que pode ser enviada para um conversor A/D e outros instrumentos geralmente está na faixa de +/- 10 volts. Sinais que se situam acima destes níveis devem ser atenuados, o que é facilmente alcançado com o emprego de divisores resistivos. Por outro lado, os sinais produzidos pela maioria dos sensores possuem valores de tensão e/ou corrente baixos, entre 1mV a 1V, tornando necessário a utilização de amplificadores que deslocam a função de transferência do sensor para níveis que possibilitam sua transmissão, posterior tratamento analógico ou digital, indicação ou gravação. Uma forma comum de condicionamento emprega um **transmissor**, que além de amplificar os sinais, pode fornecer isolação, linearização, compensação de junção fria e conversão tensão/corrente (geralmente de 4 a 20mA). A transmissão de corrente em lugar de tensão permite que sinais sejam transmitidos até 1500 metros sem perda de precisão.

De todas as funções do condicionamento, a filtragem é a mais utilizada. Ela consiste na separação de sinais indesejáveis do sinal principal. Entre os sinais indesejáveis pode-se incluir interferência da linha elétrica (60Hz), de estações de rádio e freqüências acima da metade da freqüência de amostragem. A filtragem também pode ser executada utilizando-se algoritmos que

operam sobre os dados digitais já no interior do computador. Esta técnica é denominada **Processamento Digital do Sinal (DSP - Digital Signal Processing)** (BURR-BROWN COMPANY, 1994, p.7-25)[3]. A computação da média de uma série de sinais adquiridos é útil para a redução de flutuações e constitui um exemplo de DSP. Este método é utilizável para a eliminação do ruído aleatório, mas é pouco eficaz na eliminação de sinais periódicos, como por exemplo o ruído de 60Hz introduzido pela linha elétrica. Para este fim geralmente é empregado um filtro analógico **passa alta** (figura 3.8a).

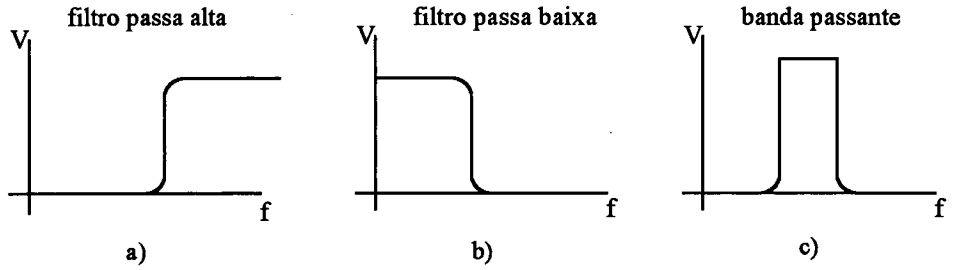


Figura 3.8 - Exemplo de filtros (BURR-BROWN COMPANY, 1994)

Os filtros, implementados em software ou hardware, são desenhados para atuar sobre tipos específicos de ruído. Em adição ao filtro passa-alta, tem-se os filtros passa-baixa e passa-banda (figura 3.8 b e c). Estes filtros podem ser fornecidos pelo fabricante junto aos terminais de ligação dos sensores. A figura 3.9 ilustra a realização eletrônica de um filtro passivo do tipo passa-baixa para um polo. Este filtro tem um corte de -6dB para um sinal de 1Hz enquanto fornece atenuação de aproximadamente 52dB (380 vezes) para um sinal de 60Hz (BURR-BROWN COMPANY, 1994, p.7-25).

---

[3] A maioria dos autores emprega o termo DSP para designar um circuito dedicado processado trabalhando em cooperação com o restante do sistema do computador.

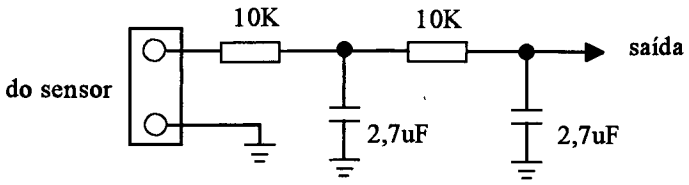


Figura 3.9 - Filtro passa-baixa (BURR-BROWN COMPANY, 1994)

Filtros mais complexos incluem a possibilidade de programação por software, constituindo subsistemas completos. Uma aplicação especial de filtros se destina à eliminação de sinais que causam interferência nos conversores A/D. Segundo o teorema de Nyquist é necessário amostrar o sinal em uma freqüência de pelo menos duas vezes a máxima freqüência de interesse na conversão. Para evitar **aliasing**, componentes do sinal com freqüência acima da freqüência de amostragem devem ser eliminados por um filtro passa-baixa. Tais filtros conseguem atenuar em até 100dB sinais indesejáveis porventura presentes na entrada do conversor.

Outra forma de condicionamento comum é a **linearização** do sinal. Muitos sensores têm uma resposta não linear sendo necessário que o resultado seja linearizado com módulos especiais em hardware e/ou software (DSP).

3.3.3.2 - Condicionamento de sinais digitais

Os sinais digitais requerem menor esforço de condicionamento, normalmente utilizando apenas adaptação em amplitude e a isolação.

A adaptação em amplitude permite que sinais com valores fora da faixa admissível de tensões para a tecnologia empregada sejam acomodados. A figura 3.10 (BURR-BROWN COMPANY, 1994, p. 3.29) ilustra um circuito simples que permite o acoplamento a uma fonte de aquisição digital onde o nível lógico alto é representado por uma tensão situada na faixa de 10

a 50 volts. A saída é compatível com a tecnologia TTL.

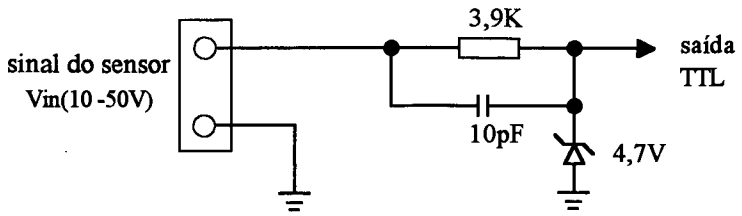


Figura 3.10 - Exemplo de adaptação digital (BURR-BROWN COMPANY, 1994)

A isolação galvânica é utilizada para separar potenciais de diferentes fontes, evitando correntes indesejáveis. Além do objetivo funcional é importante para a segurança humana, evitando que o equipamento adquira um potencial elevado em relação à terra em determinadas condições. A figura 3.11 ilustra um dispositivo isolador utilizando um acoplador opto-eletrônico. Este tipo de dispositivo alcança isolamento de cerca de 4000 volts.

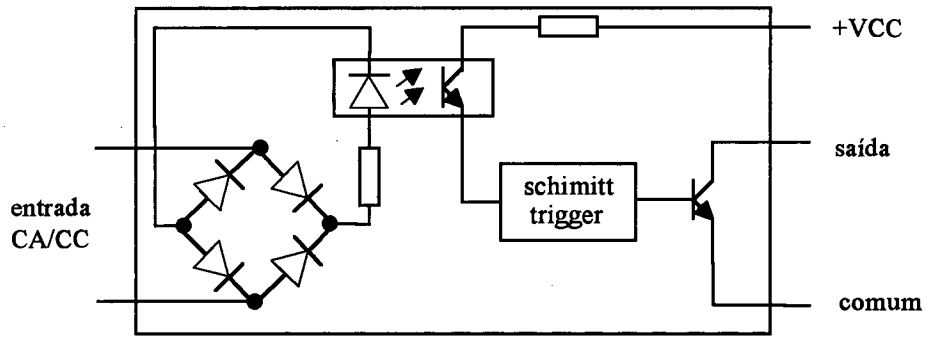


Figura 3.11 - Isolador de entrada foto-acoplado

3.3.5 - Adaptação

O computador opera apenas com sinais digitais. Após a etapa de condicionamento o sinal proveniente de uma fonte digital já está pronto para ser entregue ao computador. Já o sinal analógico necessita de sua transformação para a forma digital.

Os itens a seguir tratam dos elementos e técnicas presentes na conversão analógico para digital.

3.3.5.1 - Sistema de conversão analógico/digital (A/D)

A figura 3.3 contém um bloco de **conversão A/D** com uma disposição de componentes normalmente encontrada em placas de aquisição disponíveis no mercado. Este arranjo é utilizado para velocidades não muito altas. Para velocidades maiores outros arranjos são necessários, como por exemplo a adição de um **Sample&Hold** para cada canal. Os itens seguintes descrevem cada um dos elementos do sistema de conversão.

3.3.5.1.1 - Multiplexação - MUX

Cada conexão proveniente dos transdutores é denominada **canal de entrada**. A multiplexação permite que mais de um canal de entrada seja servido por um amplificador de ganho e conversor A/D. O arranjo mostrado na figura 3.12 ilustra o uso de um multiplexador, que é apenas uma chave selecionadora da entrada desejada. Um programa controla o multiplexador de tal forma que um canal seja selecionado para medir um sinal em um dado momento. Este sistema permite que o custo da aquisição caia consideravelmente.

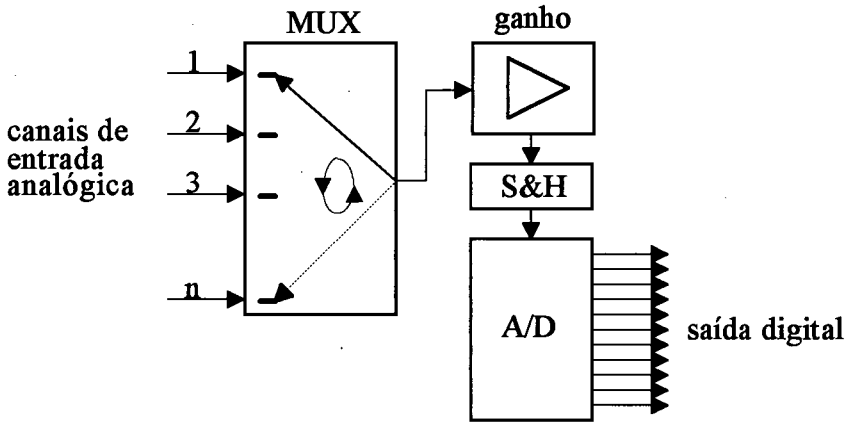


Figura 3.12 - Aquisição multiplexada

Como o amplificador e o conversor são compartilhados, a velocidade da aquisição é menor. Para um conversor com velocidade de amostragem de 100KHz recebendo sinais de 10 canais obtém-se uma taxa de amostragem por canal inferior a 10KHz. A taxa de amostragem

determina a freqüência de amostragem. Uma taxa mais alta determina a aquisição de mais pontos em um dado momento e fornece uma melhor representação do sinal original.

Outro arranjo possível para a multiplexação é denominada **arquitetura Sample&Hold simultânea**, onde cada canal é dotado de seu próprio amplificador e unidade **Sample&Hold**, aumentando a taxa de aquisição de sinais.

3.3.5.1.2 - Amplificador de ganho programável (PGA)

Permite o aumento dinâmico da escala efetiva do sistema. Através de software é possível escolher um ganho que melhor se adapte ao sinal sob medição. Comumente são encontrados dispositivos com ganhos selecionáveis de 1, 2, 4 e 8 ou 1, 10, 100 e 200, permitindo a construção de software com auto escalonamento.

3.3.3.1.3 - Sample & Hold

Em geral, um sinal analógico varia em relação ao tempo. Os conversores A/D requerem que o sinal não varie durante o ciclo de conversão. Para que isto seja possível é utilizado o dispositivo denominado **Sample&Hold** que tem por função capturar e "congelar" o valor da entrada do conversor (fig. 3.13). Este valor é mantido constante até o fim do ciclo de conversão, mesmo que o sinal de entrada sofra variação. Outra função do **Sample&Hold** é armazenar o valor do multiplexador enquanto este é chaveado para o próximo canal a ser convertido.

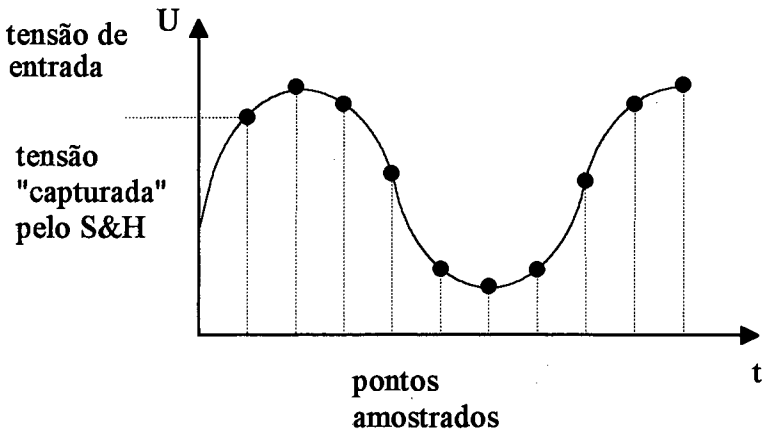


Figura 3.13 - Sample & Hold

3.3.3.1.4 - Conversor A/D

A conversão do sinal de analógico para a forma digital é efetuada pelo **conversor A/D**. Existem diversos tipos de conversores, entre estes os básicos são:

- Por aproximação sucessiva - É o mais utilizado. Consegue trabalhar com velocidades máximas de amostragem de 50KHz a 1MHz. A amplitude do sinal de entrada é comparado de forma sucessiva com valores de referência gerados no conversor. Cada valor destes corresponde a um bit no código de saída. A melhor aproximação fornece a medição. O tempo de conversão aumenta com o número de comparações

$$T = \frac{U_{\max}}{U_{LSB}}$$

onde U_{max} é a tensão máxima sendo convertida e U_{LSB} a tensão de referência do dígito menos significativo. Pelo exposto, conclui-se que o tempo de conversão é inversamente proporcional a precisão

desejada. Neste tipo de conversor é necessário que o sinal de entrada seja mantido constante durante a conversão.

- Integração - Operam na faixa de 10 a 500Hz. O conversor integra a tensão de entrada *Vx* por um período especificado de tempo *T1*. O resultado *e1* é novamente integrado de volta até zero por uma voltagem de referência *Vref*. O tempo resultante *T2* é proporcional a *Vx*

$$Vx = Vref \times \frac{T2}{T1}$$

mantendo-se o período de integração *T1* múltiplo da freqüência da rede, consegue-se uma boa rejeição deste ruído. Isto, aliado a melhor linearidade e precisão alcançada em relação ao sistema por aproximação, torna este conversor uma boa escolha para sinais de baixa freqüência, como os fornecidos por termopares.

- Conversão paralela (flash converter) - Fornecem taxas de conversão de até 100MHz. Seu funcionamento é baseado na utilização de um número *N* de comparadores em paralelo com um nível de referência para cada um. Para uma resolução de 8-bit são utilizados 2^8-1 = 255 comparadores. Para uma resolução de 10-bit são necessários 1023 comparadores. Acima desta resolução a construção se torna difícil, razão pela qual não são utilizados. O tempo de conversão depende apenas das características dinâmicas dos comparadores, não crescendo em função de seu número.

Pelo exposto, após a conversão o valor digital é apresentado por um número *Z* de bits, proporcional a tensão de entrada *Ue*,

$$Z = \frac{U_e}{U_{LSB}}$$

onde U_{LSB} é a tensão para o bit menos significativo, ou seja, Z=1 (figura 3.14).

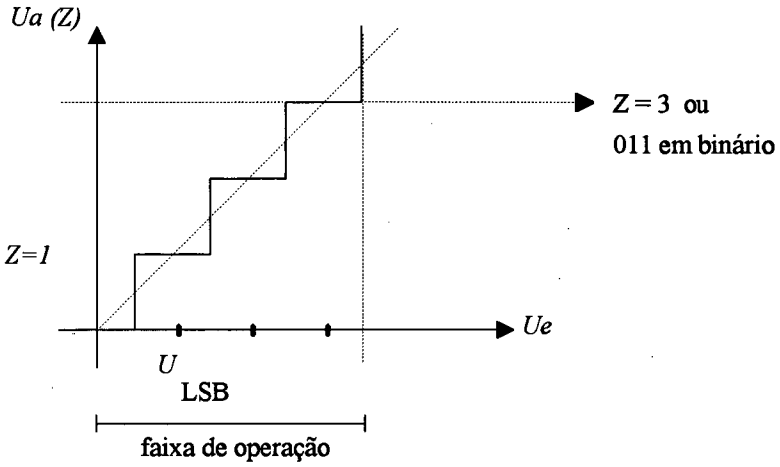


Figura 3.14 - Quantização

Cada degrau do gráfico da figura 3.14 representa um nível de tensão. Quanto maior o número de degraus para uma mesma tensão de entrada (número de bits na representação digital ou **resolução digital**), maior a precisão da conversão. São encontrados normalmente tamanhos de 8, 12, 14, e 16 bits.

Além da resolução, a largura de faixa da aquisição determina o menor e o maior valor de tensão que o conversor A/D pode quantizar. Normalmente os dispositivos comerciais de aquisição permitem a seleção de alguns níveis de tensão e ganhos, tornando possível escolher o ajuste de aquisição que melhor se adapte à resolução disponível.

A faixa, resolução e ganho determinam a menor mudança de nível detectável. Esta mudança equivale ao bit LSB (**Least Significant Bit** - bit menos significativo), e é comumente chamado de **largura de código**. O código ideal é calculado pela divisão da faixa de tensão pelo ganho multiplicado por 2, elevado à resolução utilizada. Com uma faixa de tensão de 0 a 10 volts e um ganho de 100 num conversor trabalhando com resolução de 16-bit tem-se a resolução teórica de um bit no valor digitalizado (NATIONAL INSTRUMENTS, 1993, p.3.12)

$$\frac{10V}{100 \times 2^{16}} = 1,5\mu V$$

Este resultado, em conjunto com a faixa de medição, é utilizado pelo computador para escalonar o valor real da tensão a partir do número binário resultante da conversão. Vários efeitos podem afetar este resultado teórico, sendo os erros de ganho e **offset** os mais comuns, sendo introduzidos na amplitude do sinal antes que este atinja o conversor e provocam a utilização de valores errados no escalonamento do código binário para o valor da tensão.

Para eliminar estes efeitos, é possível utilizar sistemas de correção tanto em **hardware** como em **software**.

3.4 - Sistema de Atuação

O fluxo de sinais e os dispositivos envolvidos na operação de introduzir informações e mudanças no meio físico é denominado **atuação.** Um sistema de atuação transforma sinais provenientes do computador em modificações sobre uma grandeza no meio físico. A figura 3.15 ilustra um conjunto de elementos de um sinal analógico (caminho à esquerda) e para a geração de um sinal digital (caminho à direita) a partir do computador em direção ao dispositivo atuador no meio físico.

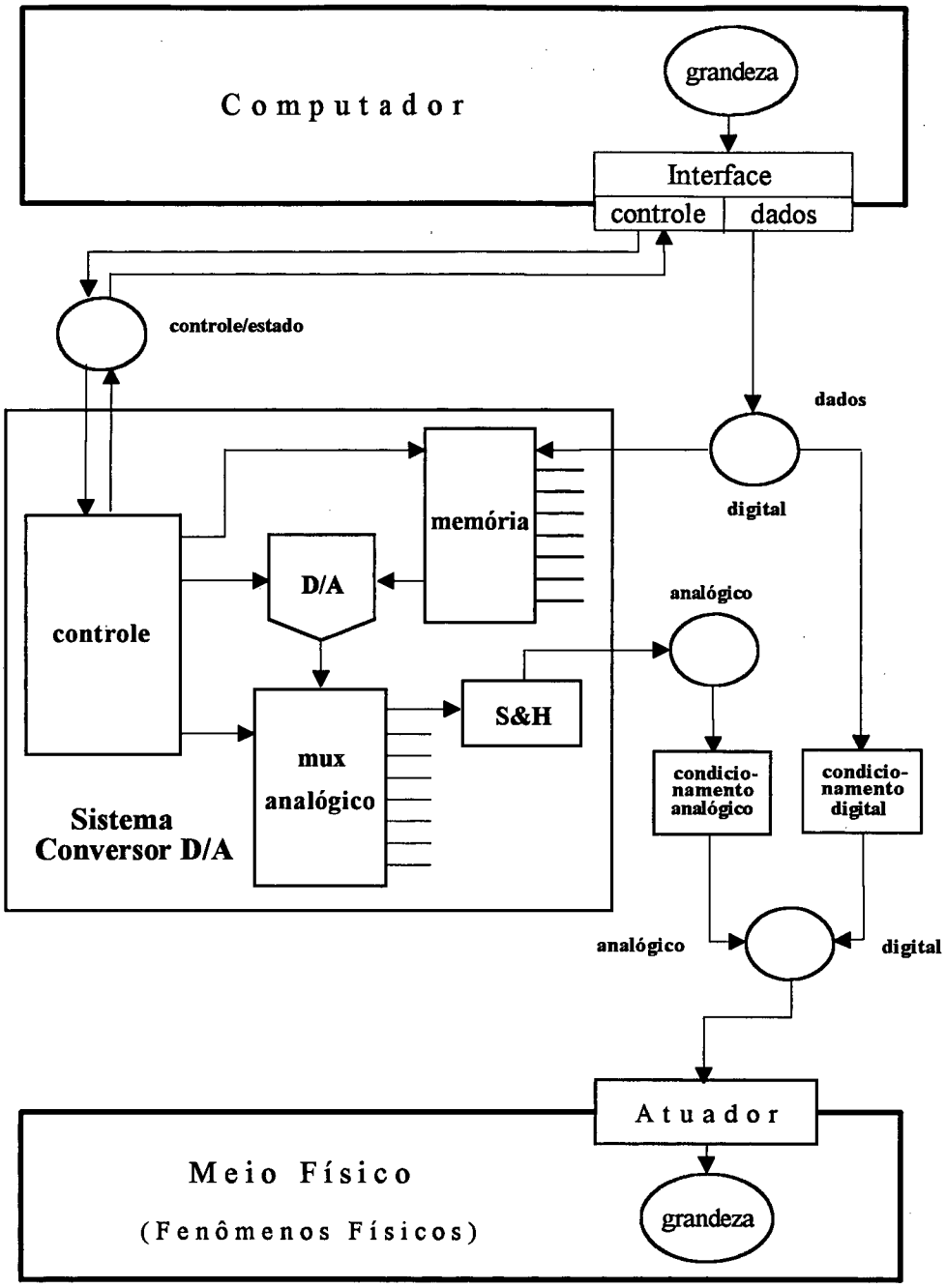


Figura 3.15 - Sistema de atuação.

3.4.1 - Adaptação do sinal

Os instrumentos atuadores geralmente precisam de corrente ou tensão elétrica em níveis maiores do que a interface de saída do computador pode oferecer. Além disto, como nos sensores, são caracterizados, sob o ponto de vista do tipo de sinal utilizado na comunicação com

o computador, em **digitais** e **analógicos** (figura 3.1). Da mesma forma, é necessário um menor esforço de adaptação para os atuadores digitais. A seguir são apresentadas as adaptações para os dois tipos.

3.4.1.1 - Adaptação para os atuadores digitais.

Muitos tipos de atuadores requerem apenas um comando de ligar ou desligar (**ON/OFF**). Neste caso a adaptação se restringe ao condicionamento, geralmente composto de isolamento e amplificação. Cada ligação de um periférico de atuação digital ao computador é feita através de um canal de comunicação denominada **porta de saída digital**. Utilizando-se a tecnologia TTL, estas saídas fornecem corrente que varia de 16 a 24 mA. Estes valores já são suficientes para a excitação de uma indicação luminosa, como um LED (diodo emissor de luz) por exemplo, ou para a ativação de sistemas que possuem em seu interior lógica compatível. Para a ativação de cargas maiores, a forma mais simples de utilização emprega um relê que, além de permitir o chaveamento de cargas elevadas, fornece isolamento. A figura 3.16 ilustra um arranjo para a ligação de um relê de baixa corrente (tipo reed switch) à interface do computador (BURR-BROWN, 1994, p.7-30).

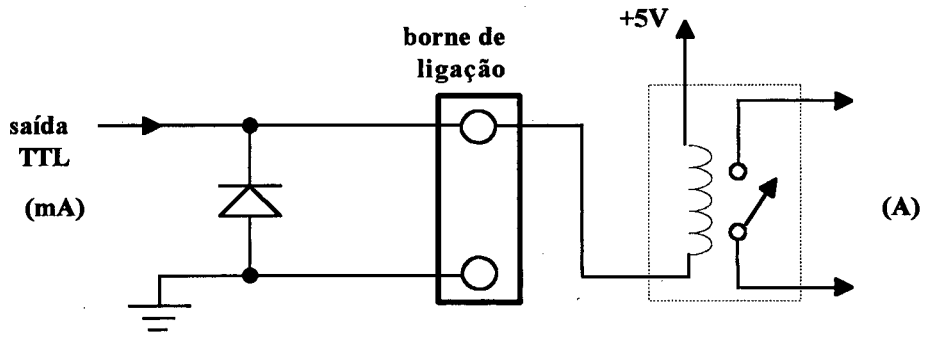


Figura 3.16 - Saída digital com relê (BURR-BROWN, 1994).

Com este circuito é possível a comutação de motores, válvulas, contadores, contatores eletromagnéticos, etc. Além do relê, é normal o emprego de elementos de estado sólido para

amplificação e comutação (triac, SCR, transistor), sendo comum o emprego de opto-acopladores como elementos de isolação galvânica.

3.4.1.2 - Adaptação para atuadores analógicos - Conversão D/A

A figura 3.15 ilustra um bloco de conversão D/A com uma disposição de componentes normalmente encontrada em sistemas de aquisição comerciais. O conversor básico ilustrado na figura 3.17 (JONES, 1977, p.127) consiste em uma referência de voltagem, um conjunto de resistores de precisão e chaves, que são comandadas pela lógica digital. Existem outras configurações mais simples e mais elaboradas, mas todas em torno do mesmo princípio.

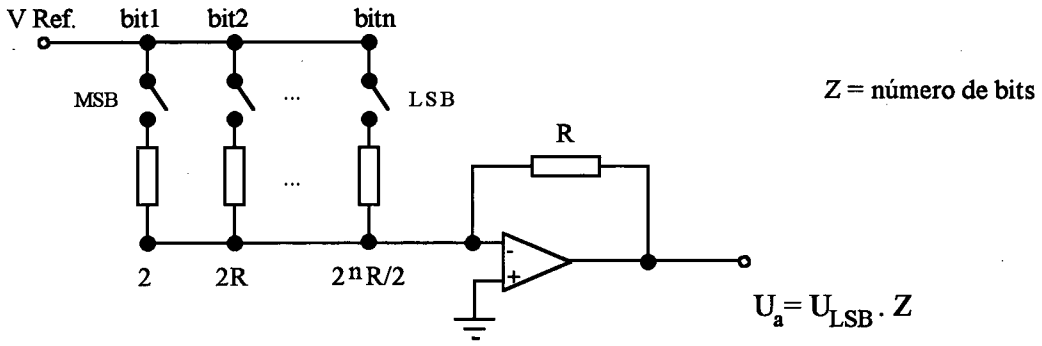


Figura 3.17 - Conversor básico D/A (JONES, 1977).

Vários parâmetros determinam a qualidade do sinal de saída produzido, entre eles o tempo de estabilização, a taxa de desvio e a resolução. O tempo de estabilização e a taxa de desvio determinam a velocidade para que o conversor DA alcance o nível necessário do sinal de saída. A resolução é similar à resolução do conversor A/D. Quanto maior o número de bits para representar um determinado valor de saída, menor a magnitude de cada incremento de tensão, resultando numa maior resolução de saída. Com uma resolução de 12-bit tem-se a possibilidade de 4096 valores diferentes. Para uma tensão de 0 a 10 volts cada bit representa um acréscimo de $2{,}441 \times 10^{-3}$ V ao valor de saída.

No conjunto conversor da figura 3.15 optou-se por uma construção que permite o uso de um conversor D/A para mais de um canal. O sinal proveniente do computador é armazenado em uma memória auxiliar, geralmente de 8 posições, que é seqüencialmente lida e seus valores transferidos para o conversor. Para isto a saída de cada célula da memória alimenta o conversor por um tempo suficiente para que se efetue a conversão. Sua saída é então transferida para um multiplexador analógico, indo para a saída correspondente ao canal endereçado na memória. Cada saída do multiplexador alimenta por sua vez um S&H, armazenando o valor analógico. O sincronismo, inicio e fim de conversão são estabelecidos pelo controlador interno. Algumas placas não possuem memória, ficando a cargo da aplicação guardar os valores a serem convertidos e sincronizar o processo. Este sistema, de forma semelhante ao utilizado nos sensores, permite uma redução de espaço e custo.

3.4.2 - Condicionamento

Após a adaptação de digital para analógico o sinal não está em condições de excitar diretamente a maioria dos atuadores. Valores comuns de saída da etapa conversora são gerados nas faixas ±5 V, ±10 V, 0-10 V e 4-20 mA. Já a entrada dos atuadores têm uma gama de valores muito variável, sendo necessário o condicionamento para cada caso.

São exemplos de funções contidas no condicionamento:

- Isolação galvânica;
- amplificação de tensão e/ou corrente, opcionalmente com ganho e offset controlados;
- conversão corrente - tensão, tensão - corrente, tensão - freqüência, ou qualquer forma de sinal necessária ao atuador;

- geração de rampa, evitando a comutação abrupta de um estado para outro, etc.

A figura 3.18 ilustra uma simplificação de um cartão condicionador (ROBERT BOSCH, 1992, p.146) utilizado para a excitação de válvulas proporcionais. A entrada pode ser acoplada diretamente à saída da etapa conversora, operando com entrada simples de 0 a 10 V ou diferencial de 0 a ±10 V . Este cartão possui ajuste manual de offset e ganho, geração de rampa e saída para alimentar o solenóide de uma válvula com consumo de 35W.

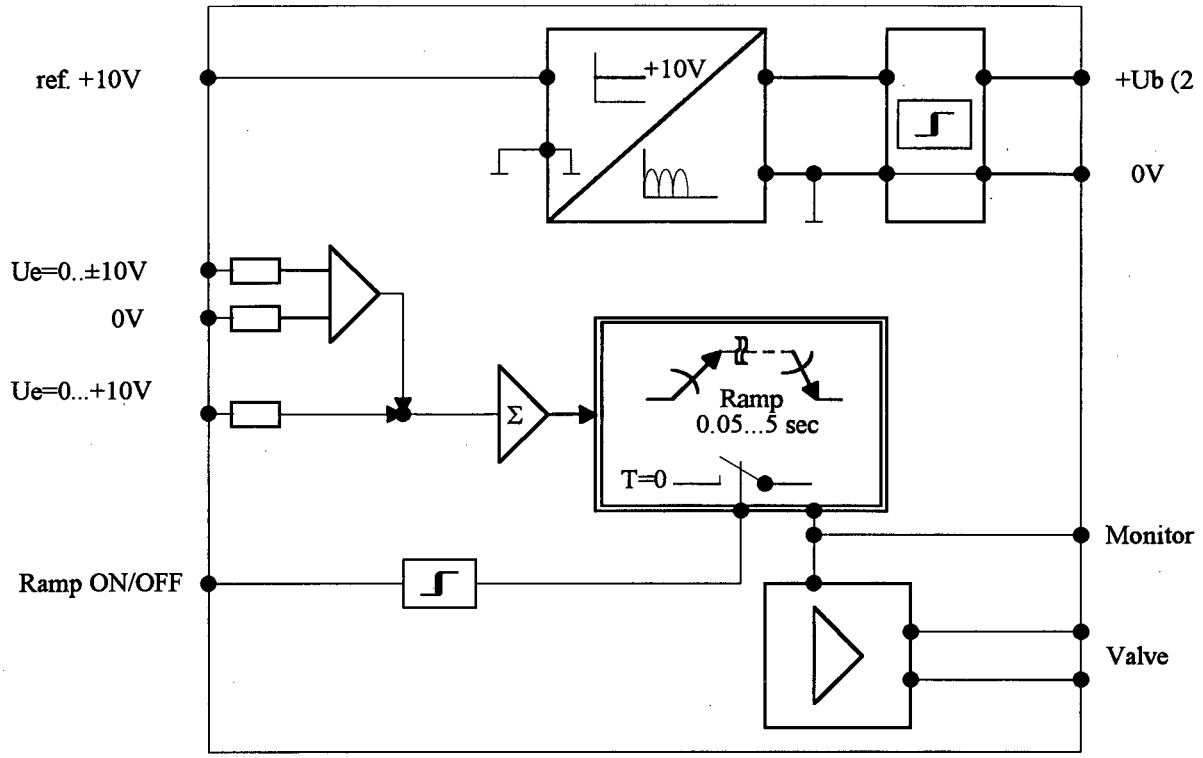


Figura 3.18 - Cartão condicionador analógico (ROBERT BOSCH, 1992)

3.4.3 - Atuadores

Os atuadores são responsáveis pela introdução de modificações em grandezas no meio físico. De acordo com o tipo de informação necessária à sua excitação, também podem ser classificados em digitais e analógicos.

3.4.3.1 - Atuadores digitais

Os atuadores que operam com informação de entrada digital geralmente recebem sinais provenientes de um canal de uma porta digital de saída (1 bit). Esta conexão permite o controle de dispositivos que possuem dois estados como ligado-desligado, direita-esquerda, avanço-recuo, etc. Uma ampla variedade de atuadores podem ser operados segundo esta característica, tais como indicadores luminosos, acionamento de motores CA ou CC, aquecedores, solenóides, válvulas direcionais, etc. A figura 3.18 ilustra o diagrama em blocos para a conexão de uma válvula direcional a um canal de E/S da interface do computador. O gráfico indica a posição da válvula em função do sinal presente na saída do canal, sendo este tipo de informação geralmente apresentado por tabelas.

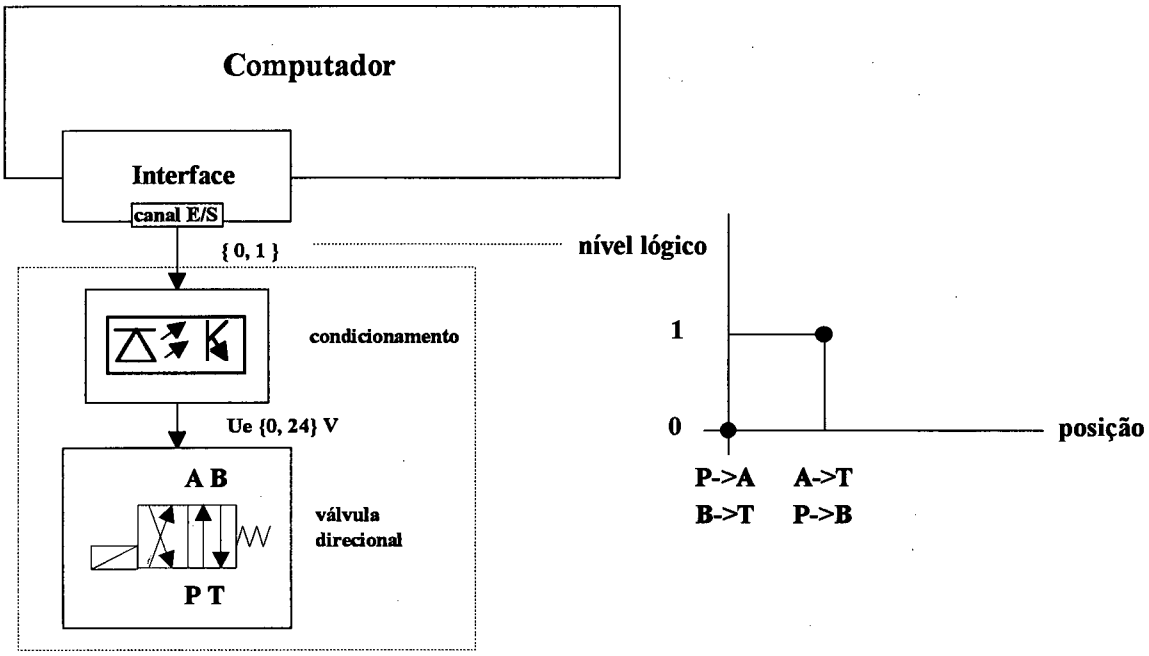


Figura 3.19 - Válvula direcional

Alguns atuadores digitais possuem um controlador adicional e se comunicam com o computador empregando mais de um bit da porta de comunicação, como no caso dos motores de passo.

3.4.3.2 - Atuadores analógicos

Entre os instrumentos atuadores que operam de maneira analógica pode-se citar servo válvulas, válvulas proporcionais, motores, indicadores analógicos, etc. Dependendo das características de construção, cada atuador possui um conjunto de valores próprio para sua excitação, daí a necessidade da existência e um condicionamento específico para cada atuador.

A figura 3.20 ilustra o diagrama de blocos para a comunicação entre computador e uma válvula proporcional para regulagem de vazão. Tomando-se como exemplo a válvula BOSCH 0811403105 (ROBERT BOSCH, 1992, p.59), é necessário uma corrente próxima de 3.7 A (25 W) para seu comando, o que não pode ser fornecido diretamente pelo computador ou conversor D/A, requerendo a utilização de um conjunto condicionador como o da figura 3.20.

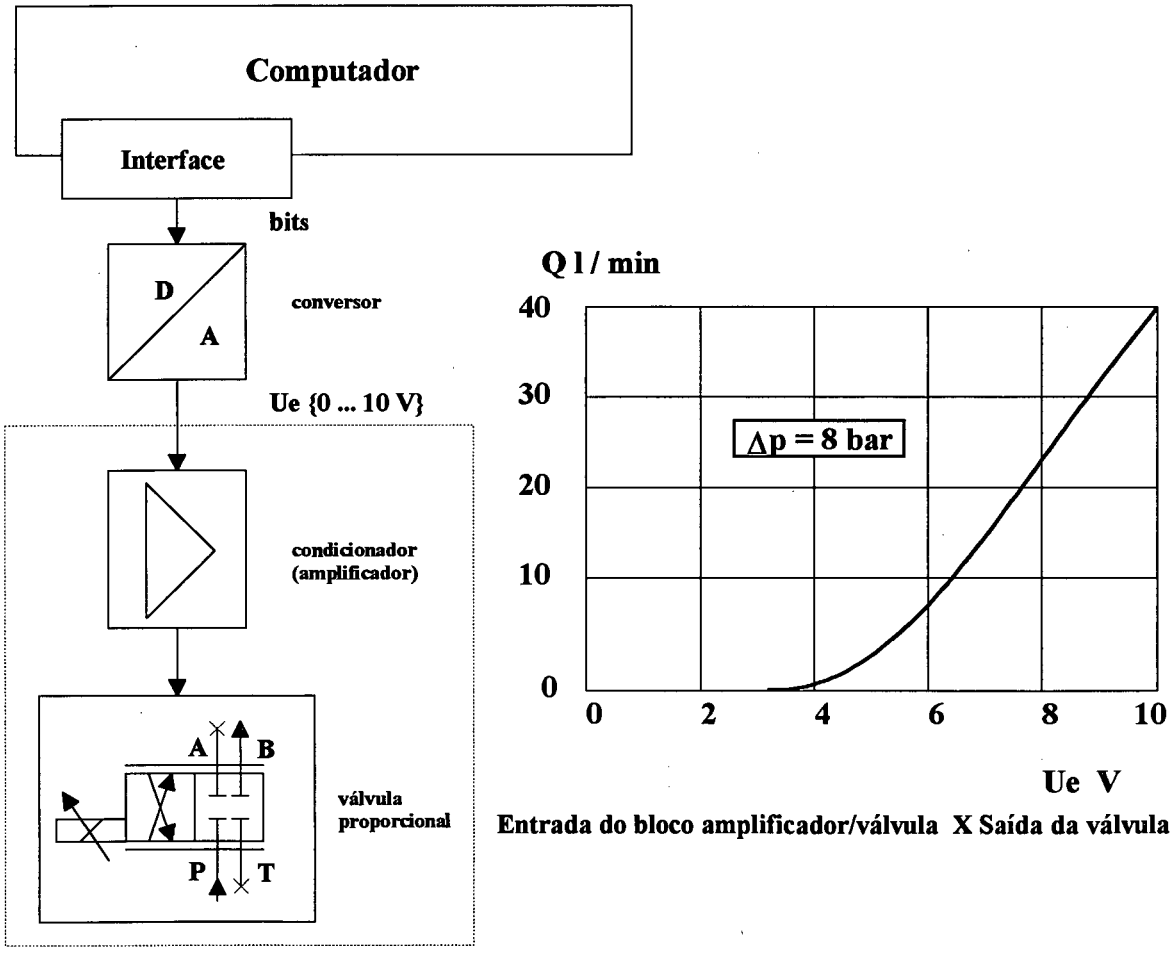


Figura 3.20 - Sistema de atuação em vazão

O gráfico apresentado na figura relaciona a vazão fornecida pela válvula e a tensão de entrada do condicionador. A não linearidade, histerese e outros ajustes na curva são geralmente solucionadas pelo programa de gerenciamento de informações no interior do computador. Neste caso específico, o fabricante não fornece a curva de resposta da válvula isoladamente, fazendo com que o conjunto condicionador-válvula seja tratado como um bloco do ponto de vista do sistema computacional.

Utilizando os processos de miniaturização eletrônica, é comum a integração de circuitos eletrônicos no corpo do dispositivo atuador para minimizar os requerimentos de adaptação. Neste caso o conjunto tem embutido uma malha fechada para controle do posicionamento do solenóide, diminuindo a histerese e aumentando a precisão do conjunto, liberando a necessidade de implementação de circuitos de malha fechada (ROBERT BOSCH, 1989, p.15). Na figura 3.21 tem-se o exemplo de uma servo-válvula com circuito controlador de posição do carretel embutido no próprio corpo da válvula fabricado pela MOOG (1994). Para este dispositivo pode-se selecionar duas formas de entrada: voltagem de 0 a ± 10V ou corrente de 0 a ± 4 mA. Ainda é possível a monitorização do estado da válvula através de uma saída para este propósito de +4 a +20 mA.

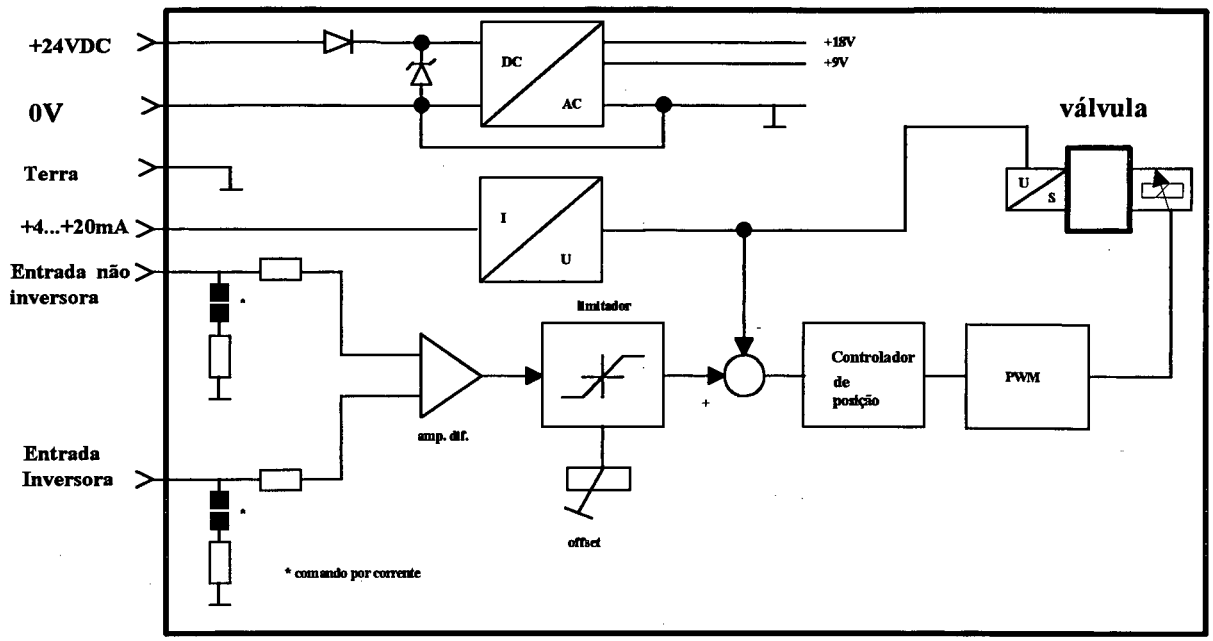


Figura 3.21 - Servo válvula (controlador embutido) (MOOG, 1994)

4 - Interfaceamento

4.1 - Introdução

Entre os sistemas de medição/atuação e o computador encontra-se uma etapa intermediária, denominada **interface**, ou mais precisamente, **interface de entrada e saída (E/S).** Este capítulo descreve a etapa da interface com ênfase nos formatos de dados e estratégias de programação.

4.2 - Características básicas

Uma interface é um dispositivo complexo que fornece o casamento entre o computador e um ou mais sistemas periféricos (figura 4.1), respeitando as características mecânicas, elétricas, formato dos dados e temporização envolvidos. Do ponto de vista do programador o papel da interface é fornecer um meio de troca de dados entre periférico e microcomputador. Como conseqüência, uma interface bem projetada deve isolar o programador de seus detalhes e se apresentar como uma "caixa preta" em que as características de E/S forneçam um simples modelo descrito por um conjunto de regras operacionais (HEWLETT-PACKARD, 1983).

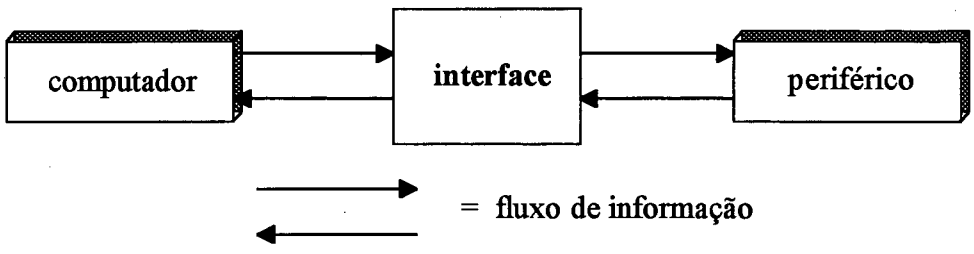


Figura 4.1 - Posicionamento da interface

Pode-se resumir em quatro as características básicas de uma interface (figura 4.2):

1. **Conexões Físicas (Compatibilidade Mecânica)** - O requisito mais simples de ser satisfeito é o casamento mecânico entre o microcomputador e o periférico. Este consiste no fornecimento do conector apropriado nos dois extremos da interface e a fiação necessária, de tal forma que as linhas elétricas de entrada, saída e controle sejam interligados corretamente. Caso a interface venha a ser utilizada como uma placa encaixada no próprio microcomputador, como por exemplo em muitas interfaces para o IBM-PC, deve ser levado em consideração as dimensões da placa e as linhas elétricas necessárias.

2. **Níveis Elétricos (Compatibilidade Elétrica)** - É necessário que características tais como níveis de tensão, níveis de corrente e freqüência de operação, sejam compatibilizados entre microcomputador e periférico.

3. **Sincronismo** - O inicio, fim e velocidade de transferência dos dados devem ser estabelecidos. Este requisito de temporização é alcançado através das linhas de controle ou estão codificados no conteúdo das mensagens trocadas. Se a velocidade do microcomputador e do periférico não são iguais, o dispositivo mais rápido deve de alguma forma diminuir sua velocidade de transmissão/recepção, permitindo a comunicação. Este mecanismo é conhecido como **handshake**.

4. **Compatibilidade no Formato dos Dados** - Uma vez que a interface tenha satisfeito os requisitos mecânicos e elétricos entre microcomputador e periférico, estes estão habilitados a trocar mensagens na forma eletrônica

através das linha de comunicação digital. No entanto os dados que são trocados entre microcomputador e periféricos geralmente necessitam de alguma forma de tradução. O microcomputador, com sua capacidade de programação, na maioria das vezes satisfaz esta função, sendo em alguns casos este papel relegado à própria interface por razões de velocidade.

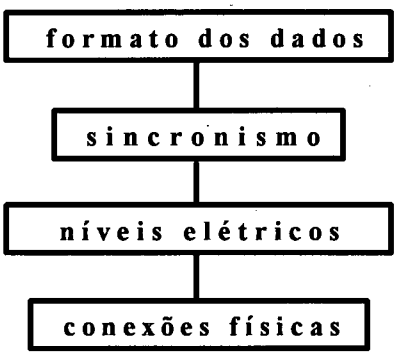


Figura 4.2 - Características básicas do interfaceamento

4.3 - Formato de dados

A transferência de dados entre computador e periférico é realizada através das **linhas de dados do computador** conectadas às **linhas de dados** da interface de E/S. Estas linhas estão em comunicação com os **registradores** da interface, que servem como local de escrita/leitura pelo programa de comunicação. Cada um destes registradores possui um endereço específico denominado **porta de E/S** ou simplesmente **porta**. A informação que se pretende transferir é escrita/lida no **registrador de dados**. Além dos registradores de dados, o mesmo caminho leva aos registradores auxiliares de **estado** e **controle**, geralmente posicionados em endereços consecutivos ao endereço do registrador de dados. O registrador de controle fornece um canal pelo qual o processador utiliza as funções programáveis do

dispositivo de E/S, e se apresenta ao computador como um registrador de escrita. O **registrador de estado** permite ao programa em execução detectar as condições ou estados envolvidos no processo de transferência, e por extensão, as condições do periférico. A figura 4.3 ilustra a disposição de ligações entre o conjunto interface e os barramentos de dados, endereço e controle do computador .

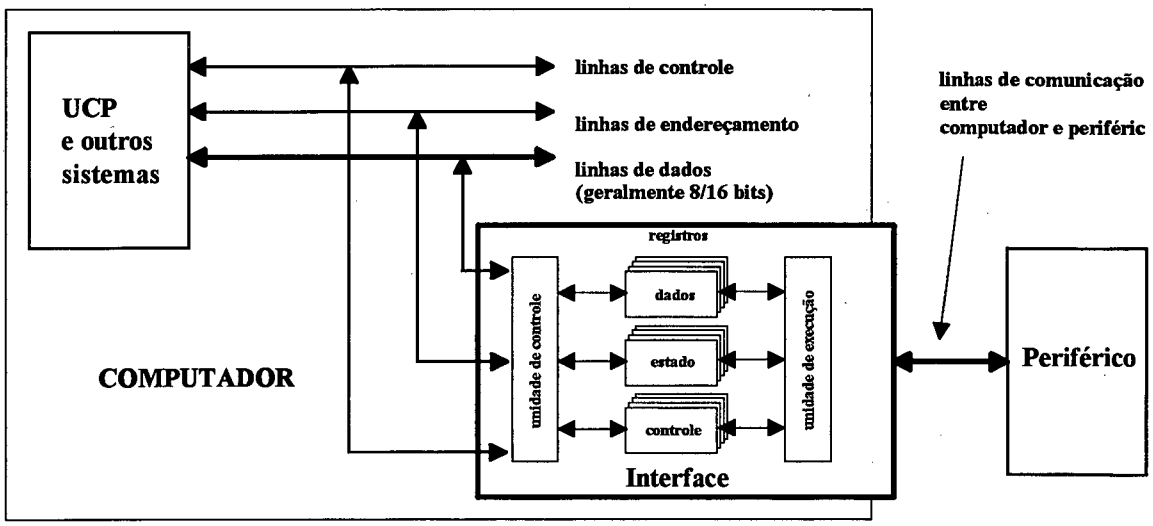


Figura 4.3 - Linhas de comunicação com a interface

Não existe restrição na interpretação dos dados enviados ou recebidos (OSBORN, 1980). Se a interface está sendo utilizada, por exemplo, para a comunicação com um conversor A/D, os bits recebidos serão provavelmente interpretados como um número binário cujo valor é proporcional à voltagem sendo medida. Quando conectado a uma impressora, os bits podem ser interpretados como um caracter a ser impresso. Do ponto de vista do dispositivo de interface, cada item de dado é simplesmente um conjunto de bits a ser enviado ou recebido. Qualquer significação ao conjunto é baseado em um nível posterior, estabelecido de acordo com o dispositivo periférico e situação empregada.

Os dispositivos de E/S possuem normalmente barramentos de dados de 8 ou 16 bits de largura. Isto envolve uma estratégia na transferência de dados de acordo com a largura do barramento de dados do computador.

No outro extremo da interface, ou seja, na conexão com o periférico, a largura de dados pode ser de um bit, caso da comunicação serial, até uma quantidade bem maior, dependente da natureza do periférico.

4.4 - Largura dos dados e estratégias de transferência

A largura dos dados depende do tipo de informação, largura do barramento de dados do periférico conectado à interface e largura do barramento de dados do computador. A figura 4.4 ilustra alguns formatos.

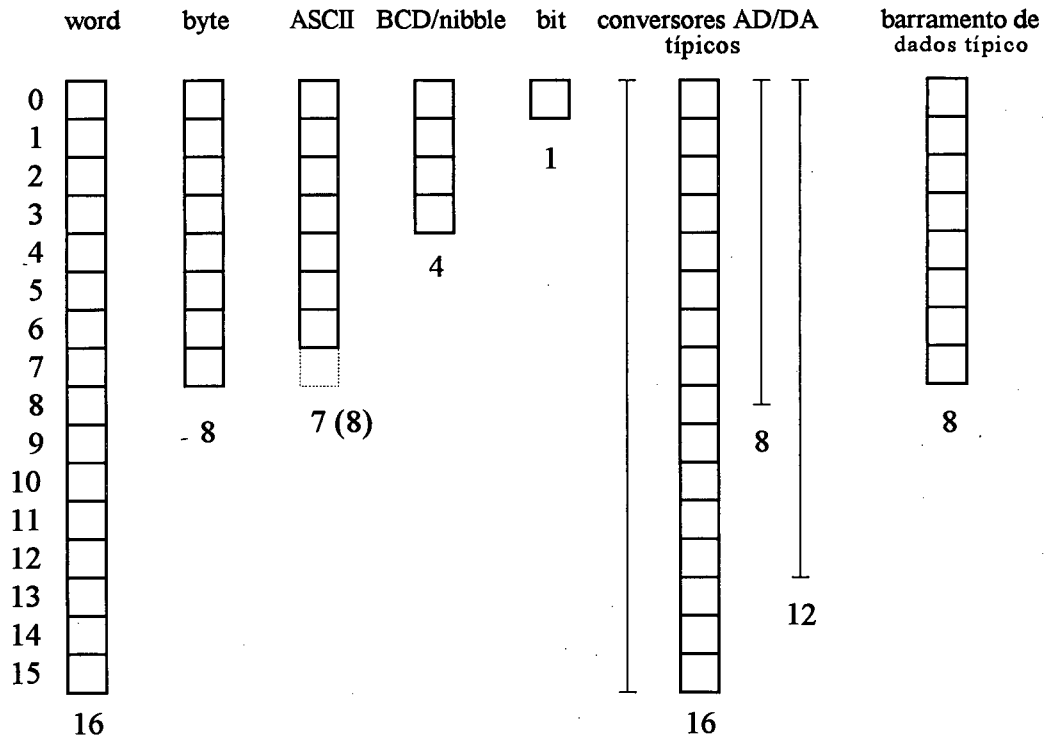


Figura 4.4 - Tamanho dos dados

Entre estes alguns tamanhos são padronizados. Para transferências de 8 bits, as estratégias são as seguintes:

- Byte (8-bit) - A leitura ou escrita da porta se dá em apenas um acesso;
- Word (16-bit) - A palavra é quebrada em um byte alto e baixo. A transferência é feita através do acesso consecutivo da porta. O dispositivo convenciona a ordem da transferência, ou seja, byte alto no primeiro acesso e byte baixo no segundo acesso, ou vice-versa;
- ASCII - A representação dos caracteres do alfabeto inglês utiliza 7 bits (0 a 6), incluindo caracteres de controle. Para permitir a acentuação e representação de caracteres especiais é adicionado um oitavo bit;
- BCD (4-bit) - A representação de números decimais permite a transferência de dois caracteres por acesso, excluindo-se o sinal, que envolve uma forma não padronizada de representação. Este formato é particularmente útil para transferências numéricas de instrumentos de medição.

A tabela 4.1 resume as formas de acesso às portas, de acordo com a largura da mesma.

Dado		Nº de acessos		Explicação
Tipo	Tamanho em bits	Para R = 8 bits	Para R = 16 bits	R= largura do barramento de dados
bit	1	1	1	acesso com o uso de máscaras
nibble	4	1	1	acesso agrupado
byte	8	1	1	representação do computador
word	16	2	1	representação do computador
dword	32	4	2	representação do computador
BCD	4	1	1	cada 4 bits representa um algarismo
ASCII	7	1	1	bit 7 pode ser usado para cálculo de paridade
EBCDIC	8	1	1	ASCII + acentos e outros caracteres gráficos
Conversores	8 /12	1/2	1	tamanhos típicos - dependente da resolução do dispositivo

Tabela 4.1 - Formas de acesso às portas

4.5 - Sincronismo

Além da diferença entre o tamanho dos dados e a largura da porta, a freqüência e o sincronismo no processo de transferência têm que ser levados em consideração. Geralmente a taxa com que são produzidos e a taxa de consumo dos dados (freqüência) não são iguais . Da mesma forma é importante o momento no tempo em que os processos de leitura e escrita se iniciam (Sincronismo).

Em sua forma mais simples este sincronismo é conseguido através da criação de informações que indicam ao outro equipamento a disponibilidade de dados. O equipamento que deseja transmitir um dado informa ao computador que este está disponível no **buffer** da sua porta de E/S. O computador por sua vez testa esta disponibilidade e colhe a informação. Em resumo, esta operação envolve duas etapas: o teste da disponibilidade da informação e a leitura do dado.

Em sentido contrário, é necessário que o computador indique ao periférico que depositou uma nova informação e que esta deve ser lida. As figuras 4.5 e 4.6 descrevem

através de redes de Petri os processos de escrita e leitura de dados em direção ao periférico, do ponto de vista do computador. As marcações em "computador" indicam a existência de nova informação no registrador de dados específico da interface em comunicação.

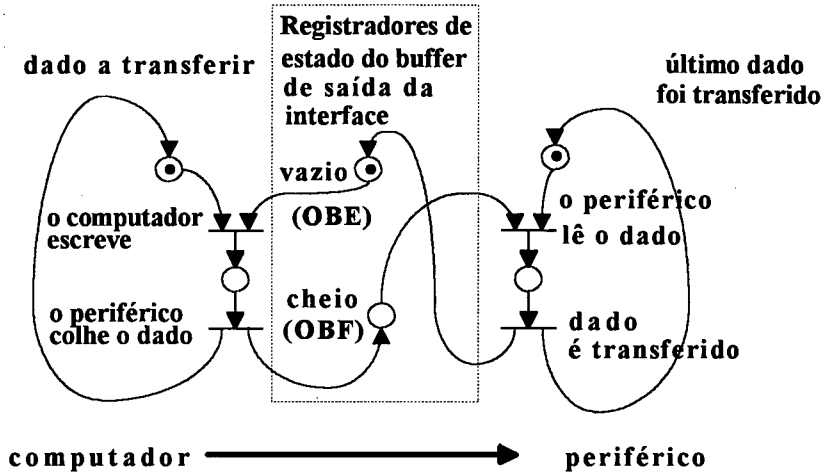


Figura 4.5 - Escrita na porta

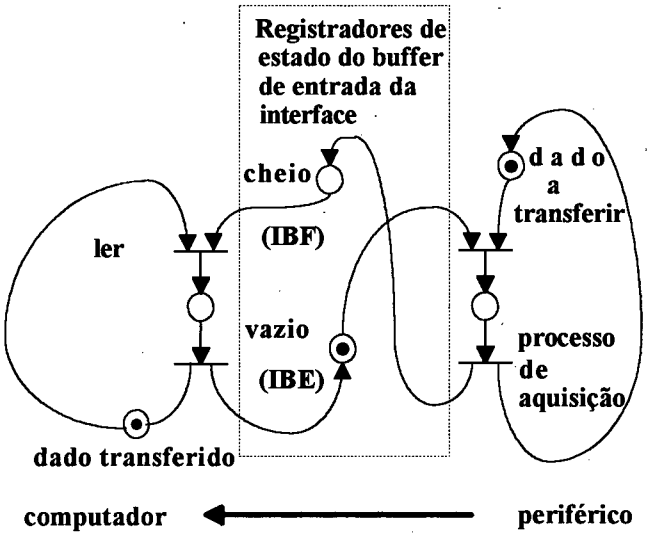


Figura 4.6 - Leitura na porta de E/S

Os registradores de estado (**status registers**) OBF, OBE, IBF e OBF são os responsáveis pela sinalização dos eventos que proporcionam o sincronismo necessário aos processo de escrita e leitura. Na transferência para o periférico (escrita), a CPU (unidade

central de processamento do computador) verifica se o periférico já colheu o dado da última escrita, verificando os bits de estado específicos, iniciando a transferência caso positivo. Este mecanismo evita que um dado seja escrito antes que o periférico tenha colhido o último dado.

Por sua vez, quando o registrador de dados recebe um novo dado proveniente do periférico (leitura), simultaneamente um bit é ligado no registrador de estado. Quando o dado é lido pela CPU este bit é desligado, informando ao periférico que um novo dado pode ser transferido. Este mecanismo impede que um dado seja lido duas vezes pelo computador.

Estes mecanismos são suficientes caso o periférico seja mais lento do que o tempo necessário para o computador efetuar a transferência e todo o processamento que esteja ocorrendo em série com este processo.

4.5.1 - Sincronismo do ponto de vista do programador

Pelo exposto, duas formas de sincronismo são necessárias - uma entre interface e periférico e outra entre a interface e o computador . Para o sincronismo entre a interface e o periférico o computador fornece os sinais **OBF** e **IBE (handshake elétrico)**. O periférico usa ou não estes sinais, não sendo verificado por programação. Entre o computador e a interface o programador pode escolher entre três métodos: **E/S programada, interrupções e acesso direto à memória**.

4.5.2 - E/S programada ou "polling"

É a forma mais simples de sincronismo, não necessitando de **hardware** adicional (HORDESKI, 1984, p.163). O programa verifica periodicamente a interface pela leitura de um ou mais registradores de estado, cujos valores permitem ao processador determinar a atividade de um dispositivo. Se o periférico necessita atenção, uma rotina de tratamento é chamada.

Caso contrário, o processador deve continuar a verificar outros dispositivos ou executar outras tarefas. Um programa baseado em E/S programada geralmente emprega um laço de execução de código simples, contendo instruções que verificam cada periférico e realizam outras tarefas. A figura 4.7 ilustra o fluxograma genérico de funcionamento da E/S programada.

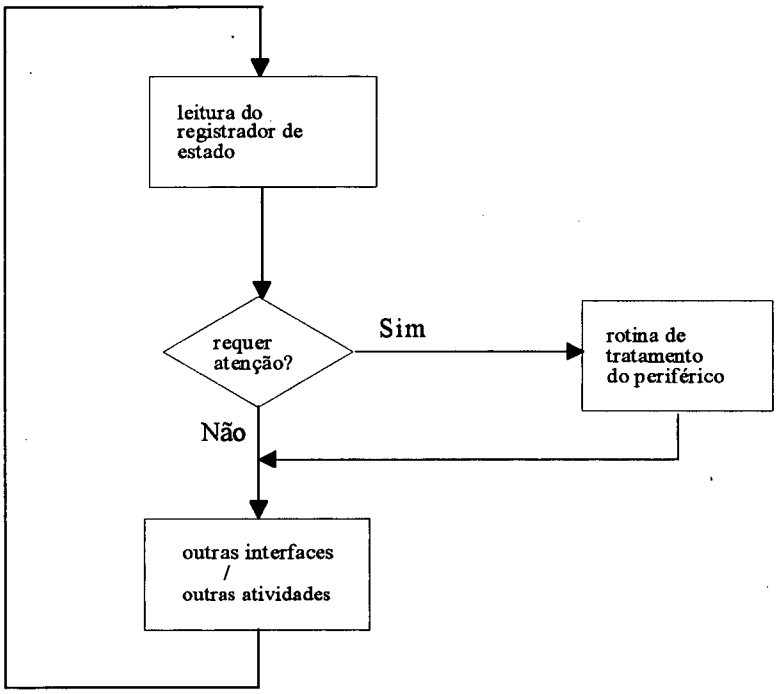


Figura 4.7 - E/S programada **(polling)**

A técnica de E/S programada é simples e de fácil implementação, porém possui algumas desvantagens. A principal reside no fato de o processador ficar preso ao laço, sacrificando o tempo de processamento dispensado a outras atividades. Em termos de desempenho, a E/S programada é a estratégia que permite grandes taxas de transferência.

4.5.3 - Interrupção

É a forma pela qual o processador interrompe a execução de um programa em andamento e passa a executar uma rotina de tratamento adequada à necessidade de atenção do

periférico. A interrupção é gerada por um sinal elétrico proveniente da interface conectada ao dispositivo periférico em direção ao processador, geralmente passando por um dispositivo de resolução de prioridades (em arranjos onde mais de uma fonte de interrupção pode ser gerada). Para a utilização de várias interrupções é necessário estabelecer uma rotina de tratamento para cada interrupção. O endereço desta rotina deve ser armazenado em uma tabela de **vetores de interrupção** de tal forma que elas passem a ser executadas automaticamente.

Na figura 4.8, o programa em andamento é interrompido no ponto 1, ao fim da última instrução em execução. Após a resolução da prioridade e a busca do vetor de interrupção (endereço do início da rotina de tratamento), o fluxo se transfere para o ponto 2. No ponto 3 a rotina de tratamento é finalizada. O ponto 4 simboliza o retorno de execução do programa principal no início da próxima instrução do programa que foi interrompido.

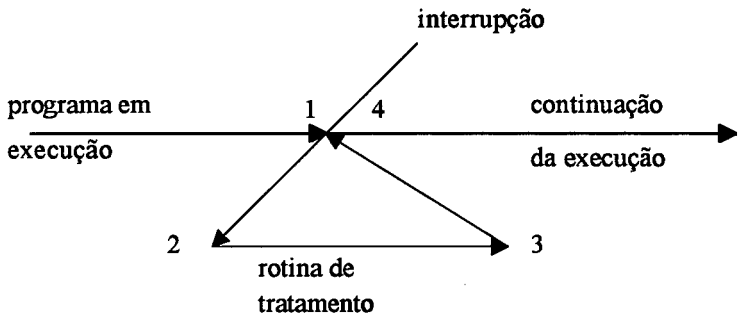


Figura 4.8 - Interrupção (OSBORN, 1980)

Fazendo-se uma análise segundo os critérios de tempo de ocupação, a interrupção alivia o processador da monitorização constante do periférico, como na E/S programada, mas acrescenta um tempo na transferência ocasionado pela troca de contexto (armazenamento e recuperação do estado corrente do processador) na chamada da rotina de tratamento. Isto torna o processo ineficiente para taxas de transferência mais elevadas.

O uso de interrupção é particularmente interessante para aplicações onde é necessária a transferência de dados em momentos específicos (temporizados). Outra vantagem é a

possibilidade de atender vários periféricos que utilizam diferentes taxas de transferência de dados.

4.5.4 - E/S com buffer

Combina as vantagens de e/s programada e interrupções. Esta técnica é particularmente eficiente para aplicações onde é necessário o emprego de laços de controle e apresentação de dados em tempo real, no qual o programa deve processar os dados a medida que são adquiridos (BURR-BROWN, 1991, p.5-3).

Uma aplicação utilizando entrada com buffer emprega um laço em que o processador espera pelos dados, processa-os e retorna para a espera de mais dados. Uma rotina de atendimento de interrupção se encarrega de armazenar os dados em um **buffer circular**, que é um vetor de tamanho moderado para o armazenamento temporário (Fig. 4.9). A rotina de tratamento de interrupção mantém um **contador de inserção** para indicar a próxima posição no buffer que é atualizada toda vez que um novo dado é armazenado no buffer. Quando o contador atinge o fim do vetor, ele é reposicionado no início, completando o círculo. Um segundo contador, indicando o ponto de leitura do vetor, é mantido pelo procedimento de E/S programada, que é chamado pelo programa aplicativo para a leitura do buffer de dados. O procedimento de E/S programada compara os dois contadores. Se o valor for diferente, o apontador de leitura indica o novo dado. O procedimento de E/S programada atualiza o contador e retorna o novo dado para a aplicação. Se não existe novo dado, o procedimento de E/S programada pode ser instruído a esperar um tempo convencionado pela aplicação ou retornar com a indicação de dado novo não disponível.

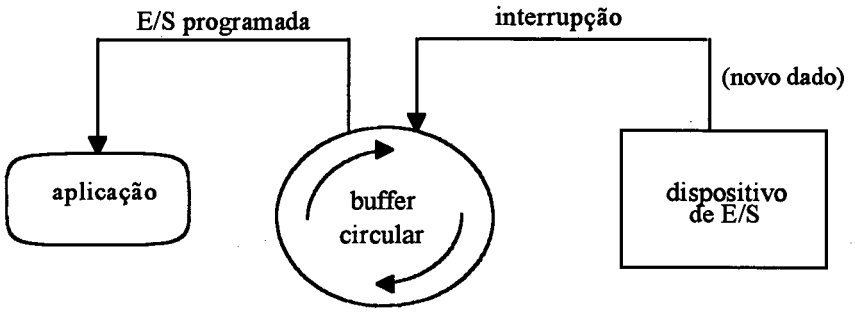


Figura 4.9 - Uso de buffer circular na aquisição

Se a taxa de transferência entre periférico e o computador for adequada ao tamanho do **buffer**, este não deve conter mais do que algumas entradas. No entanto, a rotina de interrupção compara o contador de remoção do **buffer** com o de inserção atualizado para detectar um possível estouro. Se isto acontece, o procedimento de **polling** deve retornar um valor de erro para a aplicação.

Um **buffer** de saída comporta-se de maneira similar. A aplicação passa dados para o procedimento de E/S programada. Os dados são armazenados no **buffer** circular se houver espaço para isto, caso contrário são rejeitados. Neste caso o procedimento de E/S programada atualiza o ponteiro de inserção e a interrupção o de remoção. Se os dados têm que ser atualizados em intervalos de tempo regulares, no caso de geradores de ondas por exemplo, não é permitido deixar o **buffer** vazio. Se os dados são enviados para uma impressora ou dispositivo similar, a interrupção deve ser desativada, caso o **buffer** se esvazie.

4.5.5 - Acesso direto à memória - DMA

Em muitas aplicações, particularmente na **aquisição de dados**, o uso de E/S programada ou interrupção não consegue fornecer uma taxa suficientemente alta. A quantidade de tempo necessário para responder a uma requisição de DMA é extremamente pequena em relação ao tempo gasto para o desvio e execução de uma interrupção ou executar

um laço de E/S programada. O emprego de DMA torna possível a execução do processo de transferência com taxas mais elevadas. Como a técnica de DMA é feita através de circuitos eletrônicos, a única limitação imposta na transferência é a largura e freqüência características do barramento.

Durante a execução normal de um programa, o processador tem o barramento do sistema sob seu domínio, fornecendo endereços e sinais de controle, sendo a origem e o destino dos dados. Quando uma interface deseja transferir dados através de DMA, ela envia um sinal de requisição para o **controlador de DMA**. O controlador resolve a prioridade da requisição e envia o sinal **hold** ao processador. Ao final do ciclo da instrução corrente, o processador se desliga do barramento do sistema e envia um sinal de reconhecimento ao controlador de DMA, indicando que o barramento está livre. O controlador de DMA então se conecta ao barramento, executando um ciclo de transferência entre a interface e a memória. A interface é notificada através do sinal de reconhecimento (**DMA acknowledge**) enviado pelo controlador. É importante observar que o controlador de DMA não manipula o dado no momento da transferência, sendo este escrito ou lido diretamente da memória pelo periférico. Ao fim do ciclo de transferência, o controle dos barramentos retorna para o processador.

Com esta técnica a transferência de dados é transparente ao programa em andamento, eliminando os problemas com trocas de contexto e dedicação da UCP. O processo de DMA "rouba" alguns ciclos do sistema, degradando um pouco seu desempenho (Fig. 4.10).

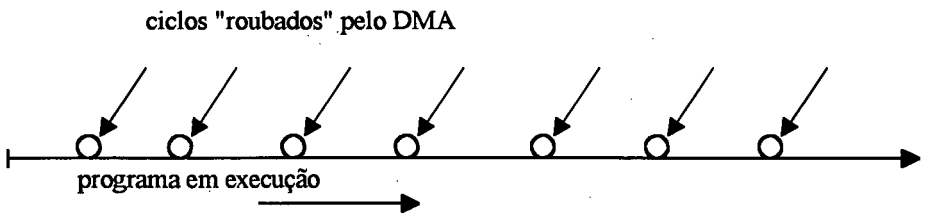


Figura 4.10 - Acesso direto à memória (DMA)(OSBORN, 1980)

A forma de programação para o emprego de DMA é simples: O programa controlador deve fornecer um endereço inicial de memória para a transferência, quantidade de dados a ser transmitida e a direção dos dados. Feito isso a transferência tem início sem intervenção do processador, caracterizando-se como uma operação em segundo plano (**background**).

O DMA também apresenta certas limitações. A principal reside no fato de ser executada apenas a transferência de um tipo de dado por canal de DMA (um computador possui geralmente mais de um canal de DMA).

4.5.6 - DMA com buffer

Uma forma melhorada de utilizar DMA é o emprego de dois **buffers** de uma maneira conhecida como **ping-pong** (fig. 4.11). Dois **buffers** são criados dinamicamente (em tempo de execução do programa). Enquanto um **buffer** é preenchido pelo DMA o conteúdo do outro **buffer** está disponível ao programa aplicativo para qualquer processamento necessário.

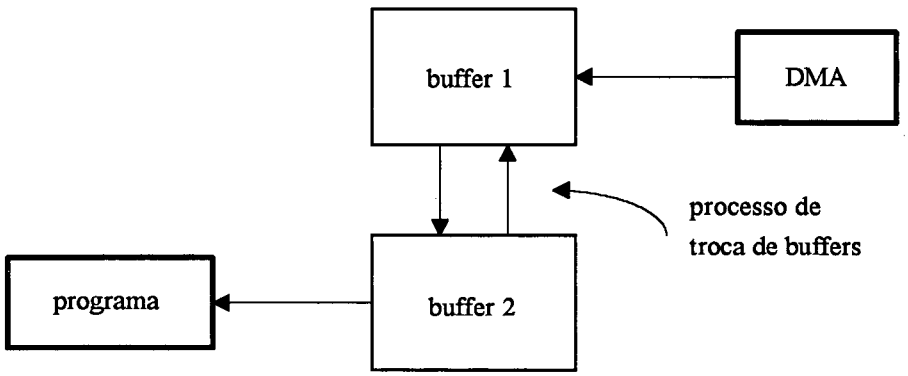


Figura 4.11 - DMA com buffer (**ping-pong**)

Quando a transferência executada sobre o primeiro **buffer** é completada, os **buffers** são trocados e uma nova transferência é iniciada. O tamanho dos **buffers** é determinado de tal forma que o programa tenha tempo de executar todo a manipulação necessário sobre os dados

no intervalo entre as trocas de **buffers**. A troca dos **buffers** se dá sob controle de uma interrupção, acionada segundo o nível de preenchimento dos **buffers** (fig. 4.12).

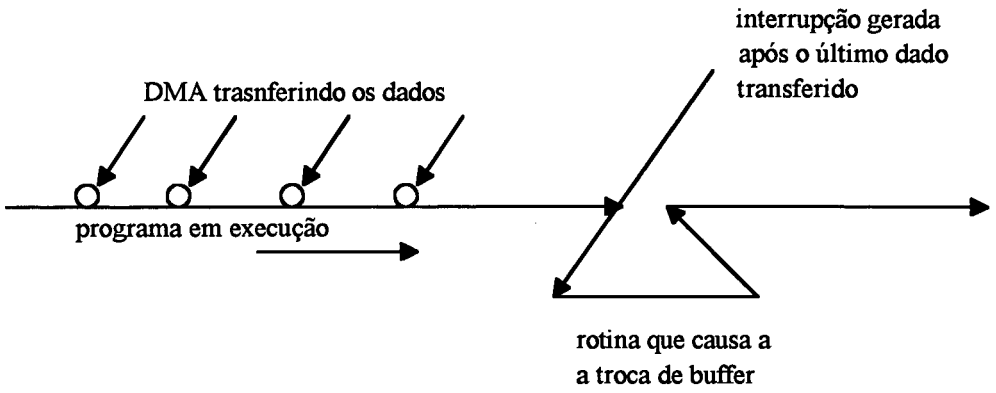


Figura 4.12 - comportamento do DMA com buffer (OSBORN, 1980)

4.6 - Interfaces padronizadas

Interfaces de uso específico, como as utilizadas em sistemas de medição e atuação, são comumente encontradas em placas encaixáveis no interior dos computadores, diretamente em seus barramentos. Tais placas possuem um protocolo de comunicação que varia de fabricante para fabricante. Para sanar este problema existem algumas interfaces padronizadas (tais como RS-232, IEEE-488, Centronics) que alivia do projetista a necessidade de se preocupar com os detalhes da interface e trabalhe com características mais próximos da aplicação.

4.6.1 - Interface Serial RS-232

A interface RS-232 é um padrão bem antigo e difundido, utilizado principalmente para conectar equipamento de processamento de dados à equipamentos de comunicação, como por exemplo terminal - MODEM. O padrão RS-232 cobre os aspectos mecânicos, elétricos e de características funcionais da interface (EGGEBRECHT, 1987, p. 200).

A RS-232 é utilizada para a comunicação até distâncias de metros com taxa de transferência de até 20 Kilobits por segundo. A quantidade de condutores mínima entre um equipamento e outro é de dois fios. Vários tipos de periféricos utilizam a RS-232, tais como impressoras, analizadores lógicos, terminais, etc.

4.6.2 - Interface IEEE-488

A sigla IEEE-488 descreve um padrão de interfaceamento digital para instrumentação programável, que desde 1975 se tornou o mais popular meio de interconexão entre instrumentos e controladores em laboratórios, testes automáticos e também aplicações industriais (NEC, 1983, p. 6-34).

A comunicação do computador com a interface IEEE-488 é geralmente feita através de uma placa encaixável diretamente aos barramentos do computador ou se utiliza de uma interface de comunicação intermediária, como por exemplo RS-232 (fig. 4.13). Os periféricos que utilizam a interface IEEE-488 são geralmente instrumentos complexos, tais como freqüencímetros, osciloscópios, analisadores de espectro, etc.

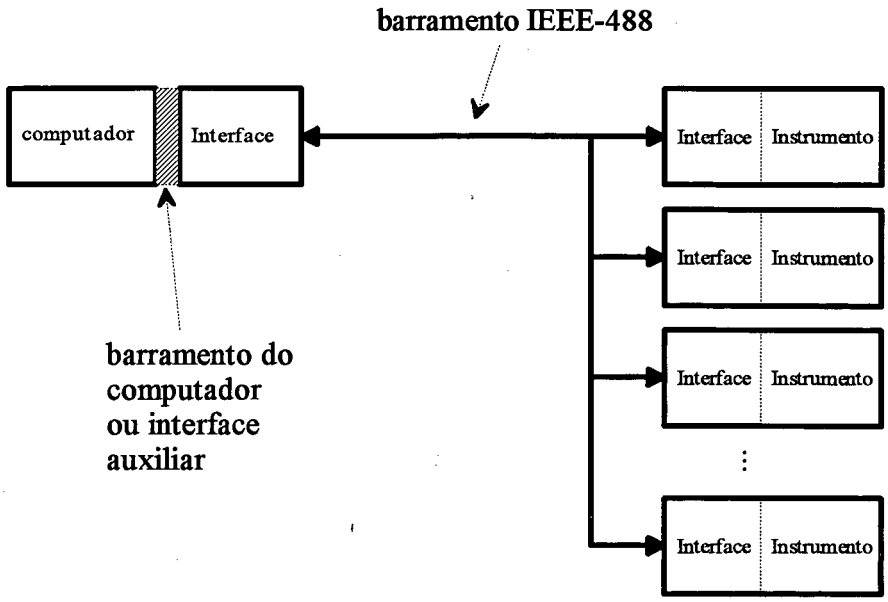


Figura 4.13 - Interfaceamento IEEE-488

O principal propósito da interface IEEE-488 é a transferência de informação entre dois ou mais dispositivos periféricos (instrumentos). É possível a conexão de até 15 dispositivos, sendo a distância máxima de 20 metros. Três tipos de dispositivos são definidos pelo padrão: originador de dados (**talker**), receptor (**listener**) e controlador (**controller**), embora alguns dispositivos possam combinar funções **talker/controller** e **talker/listener**. A configuração mínima requer pelo menos a presença de um dispositivo com o papel de controlador com a função de determinar qual dispositivo irá se comunicar com outro qualquer. Cada dispositivo recebe um endereço, que é um número de 0 a 34.

A conexão entre um dispositivo e outro é feita com um cabo de 24 vias, sendo 16 para a transmissão de sinais e oito para aterramento (fig. 4.14). A conexão das linhas aos instrumentos é feita através de um conector padronizado para a interface. As vias são agrupadas em três barramentos:

- **Dados** - Barramento por onde trafegam as informações de equipamento para equipamento. Tem a largura de oito vias e é

compatível com o código ASCII. A transmissão dos dados tem a forma bit-paralelo, onde cada byte é transferido de maneira serial.

- **Controle de recepção/transmissão** - Fornece o **handshake,** estabelecendo a comunicação entre as interfaces antes que um código possa ser enviado pelo barramento de dados. As linhas de controle são três:

  NFRD -not ready for data

  DAV - data valid

  NDAC - not data accept

- **Controle** - 5 linhas especiais de controle dos instrumentos, como endereçamento de dispositivos, geração de interrupção etc. :

  ATN - attention

  SRQ - service request

  IFC - interface clear

  EOI - end or identify

  REN - remote enable

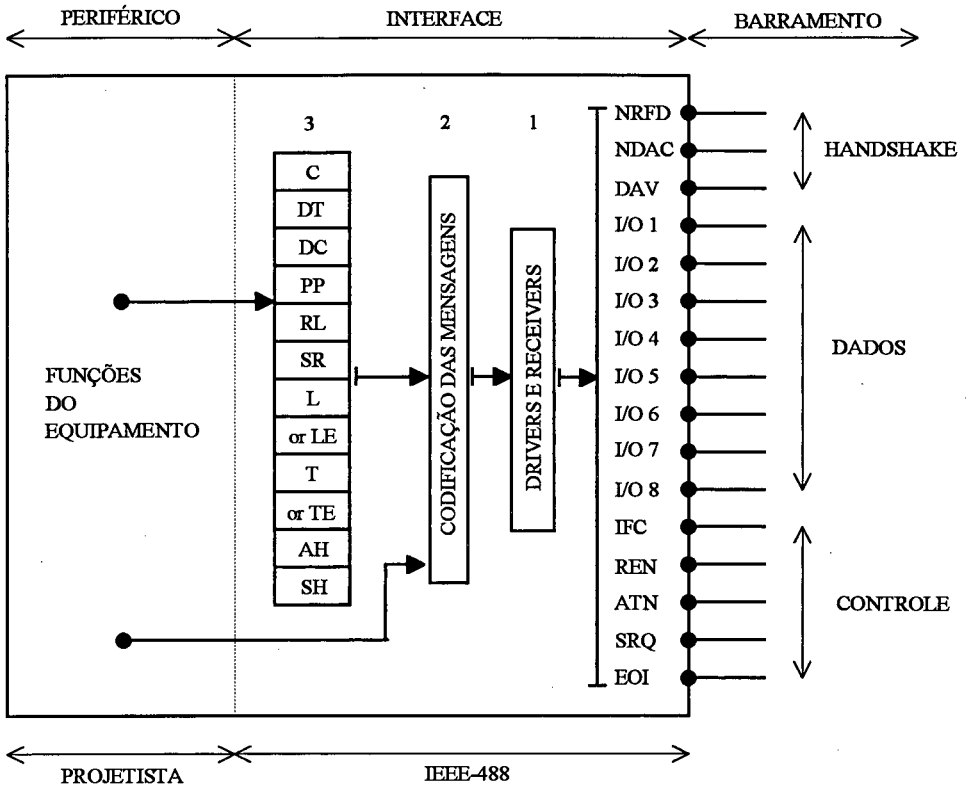


Figura 4.14 - Funcionalidade da interface IEEE-488

A figura 4.14 (GILBERT, 1982) ilustra as três responsabilidades gerais da interface:

1. Fornece o interfaceamento elétrico, responsável por manter dentro dos requisitos do padrão IEEE-488 os níveis elétricos;

2. Fornece ao barramento um padrão específico de níveis elétricos altos e baixos (1 e 0) de acordo com o caracter ou código de controle a ser enviado pelo barramento.

3. Conjunto das 10 funções de interfaceamento do padrão IEEE-488. Através destas funções o projetista estabelece a comunicação entre os dispositivos.

Antes de uma seqüência de comunicação possa ser iniciada é necessário especificar qual dispositivo assume o papel de transmissor dos dados (**talk**) e quais os dispositivos que atuarão como receptores (**listen**). A função de delegar os papéis é conduzida pelo dispositivo **controlador ativo (Active Controller)**, já que existe a possibilidade de mais de um dispositivo controlador.

Para estabelecer a comunicação entre um dispositivo transmissor e outros receptores, o **controlador ativo** poderia, como exemplo, efetuar os seguintes passos:

1. Ajusta a linha **ATN (Attention)** para encerrar qualquer transferência de dados sendo executada;

2. Desfaz o papel de receptor de todos os outros dispositivos;

3. Designa qual dispositivo será o transmissor;

4. Determina quais dispositivos atuarão como receptores;

5. Indica a todos os dispositivos que o processo de transferência pode ser iniciado, retirando o sinal de atenção (**ATN**) do barramento.

Uma vez que a transferência tenha sido iniciada, o dispositivo com o papel de transmissor espera até que os receptores estejam prontos para receber e então coloca o primeiro dado (**byte**) no barramento. Após sinalizar que o dado presente é válido, ele aguarda que todos os receptores tenham aceitado o dado. Após aceito, o transmissor coloca o próximo dado no barramento e espera que todos os receptores estejam prontos novamente. Após isto o transmissor indica que o dado é válido e a seqüência se repete até que toda a transferência tenha ocorrido.

4.7 - Interfaceamento em placas de aquisição

O computador além de receber os dados adaptados do sistema de aquisição também atuam neste processo em maior ou menor grau, de acordo com a complexidade do sistema de aquisição. De um modo geral o computador se comunica escrevendo em registradores de controle da interface do sistema de aquisição e controla os seguintes fatores:

- **Modo de transferência** - pode-se ajustar o modo com que a transferência dos dados se dá entre o sistema de aquisição e computador (interrupção, E/S programada ou **DMA**);
- **Configuração** - Ajusta-se o modo de entrada (diferencial ou simples), faixa de entrada, polaridade da entrada, etc.;
- **Limpeza** - zera o circuito de entrada analógica e memórias auxiliares, etc.;
- **Canal** - escolha do canal, grupo de canais ou seqüência cuja medição interessa em um determinado instante (MUX);
- **Ganho** - ganho estabelecido para aquele canal (PGA);
- **Inicio de conversão** - momento em que a conversão será iniciada, logo após a seleção do canal e ganhos desejados.

Para que o computador possa avaliar eventos da conversão, a interface fornece registradores de estado (como por exemplo, no modo de operação por E/S programada, onde registrador de estado fornece a informação de que o **fim do ciclo de conversão** foi alcançado).

Além das linhas de dados, a interface também se conecta às linhas de endereçamento de controle. Através das linhas de endereçamento da interface é possível selecionar o registrador interno desta que irá receber ou enviar os dados.

4.8 - Visão resumida

Pelo exposto nos capítulos anteriores, a medição e atuação envolve, além dos elementos periféricos, tais como conversores, condicionadores, multiplexadores, o uso de um elemento interfaceador e de rotinas para o estabelecimento do protocolo de comunicação. Na fig. 4.15 observa-se como um todo os elementos principais de um sistema em relação ao interfaceamento:

1. A aplicação sendo executada no computador;

2. As rotinas de comunicação com as interfaces, estabelecendo o protocolo de comunicação e armazenamento dos dados;

3. A interface, que compatibiliza os sinais provenientes e direcionados para os periféricos com os sinais do computador;

4. O periférico.

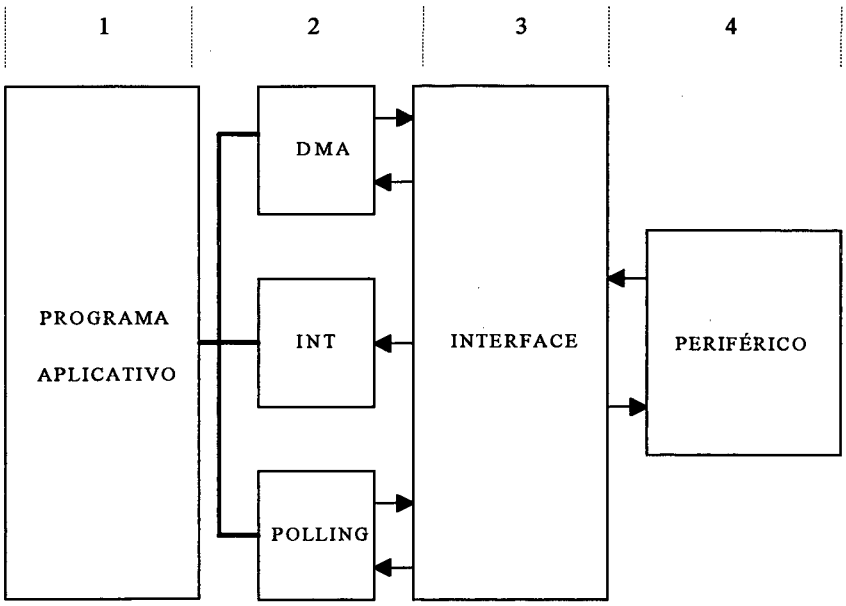


Figura 4.15 - Elementos da comunicação (EGGBRECHT, 1987)

5 - Orientação por Objetos

5.1 - Introdução

A **orientação por objetos** é uma nova forma solucionar problemas na **engenharia de software,** utilizando modelos organizados ao redor de elementos e processos presentes no mundo real. O elemento construtivo fundamental é a classe/objeto, que combina em uma única entidade, estrutura de dados e comportamento (RUMBAUGH et al., 1991, p.1).

O orientação por objetos utiliza fundamentalmente os mesmos elementos tradicionais na construção de sistemas computacionais: dados e procedimentos. No entanto, enquanto no processo tradicional de desenvolvimento a atividade está na identificação de procedimentos, a orientação por objetos focaliza o **encapsulamento de dados + procedimentos** como elemento fundamental. Em adição, a análise e projeto detalhado não se baseiam apenas nestes objetos, mas também nos serviços oferecidos por estes. Através da utilização de mensagens, os objetos passam informações, evocam serviços de outros objetos e implementam processos (HENDERSON-SELLERS & EDWARDS, 1990, p. 146).

O projeto detalhado, incluindo codificação de procedimentos e especificação de estrutura de dados, é adiado até bem tarde no processo de desenvolvimento, e são detalhes privados dos elementos (classes/objetos), indo de encontro às necessidades de **information hiding** (PARNAS, 1972). Por conseguinte, os procedimentos algorítmicos e dados não estão especificados de maneira rígida em um nível alto de abstração no desenvolvimento, tornando um sistema orientado por objetos mais flexível, uma vez que mudanças no nível de codificação são mais facilmente absorvidas sem a necessidade de mudanças no projeto. Os tópicos a seguir expõem os principais elementos sobre engenharia de software baseados no paradigma orientado por objetos e também aspectos do processo de desenvolvimento de software (**life-cycle**).

5.2 - Decomposição

O caminho para lidar com a complexidade de sistemas é a decomposição. A decomposição busca a divisão de um problema em partes pequenas o suficiente para que as interações entre elas sejam simples e bem definidas. Desta maneira os elementos de um sistema podem ser tratados de forma independente e assim mesmo manter o grau de comunicação necessário (LISKOV, 1986, p.5). A habilidade de modelar um sistema como uma hierarquia de sub-modelos é essencial na redução da complexidade. O uso destes sub-modelos permite a verificação individual de cada componente antes da montagem do sistema inteiro. Em adição, a estrutura do modelo é mais inteligível, facilitando sua documentação e modificação.

5.3 - Abstração

A abstração[4] é o processo de focalizar apenas as características que interessam de um determinado elemento dentro de um contexto específico, desprezando os detalhes que não contribuem para sua descrição dentro deste contexto. Isto permite simplificar a análise e descrição do problema, separando os atributos que são relevantes para a solução.

A abstração depende do nível a partir do qual é estabelecido a visão dos elementos do problema. Os detalhes que não precisam estar presentes neste nível são omitidos, podendo aparecer em outros ângulos de visão do problema. O resultado deste processo é a criação de um **modelo**, que é a abstração sob aspectos específicos.

5.4 - Decomposição X Abstração

A combinação da decomposição com a abstração é fundamental no projeto de um sistema computacional: a decomposição é utilizada para "quebrar" o problema em componentes que podem ser combinados para resolver o problema original; a abstração permite a escolha correta

---

[4] Abstração: ato de abstrair. Abstrair: Abster-se de considerar uma ou mais propriedades separadas mentalmente de um todo concreto ou intuitivo. **Dicionário Aurélio Básico da Língua Portuguesa**. 1988.

dos componentes. A alternância entre estas duas técnicas prossegue até que o problema original seja reduzido a um conjunto de problemas de fácil resolução.

5.5 - Tipos de abstração

Dentro de uma solução computacional são comuns dois tipos básicos de abstração:

- Abstração por procedimentos - Permite a extensão da máquina virtual definida por uma linguagem de programação através da adição de novas operações. É a base da decomposição clássica (funcional) de um problema. Nesta solução existe uma função (procedimento) principal a partir da qual outras funções (procedimentos) são evocados (**top-down design**);

- Abstração de dados - Permite a extensão através da definição de novos tipos de dados. É a base do desenvolvimento orientado por objetos. Nesta solução o programa que manipula os objetos pode ser deixado como o último elemento a ser definido (**bottom-up design**).

Segundo ECKEL (1989, p4) a orientação por objetos é uma nova forma de pensar sobre a resolução de problemas através do computador, onde em lugar de se adaptar o problema a algo familiar ao computador, o computador é adaptado ao problema. Na orientação por objetos o problema é examinado a procura de "entidades" que se relacionam com outras partes do problema. No método tradicional o problema é artificialmente decomposto em funções e dados, adaptando-o à arquitetura de **hardware** von Neumann (JACOBSEN, 1992, p. 74).

5.6 - Tipo de dado abstrato

Um **tipo de dados abstrato** consiste em uma representação de dados (estrutura de dados) e um conjunto de operações que atuam sobre estes dados,

tipo de dados abstrato = { estrutura de dados, operações }.

O acesso aos dados só é possível através das operações definidas para o tipo abstrato, o que é denominado **encapsulamento**. O encapsulamento é alcançado através de regras de acesso embutidas nas linguagens que implementam o tipo abstrato (ADA, Modula 2, linguagens orientadas por objeto).

O elemento básico de encapsulamento compreende um bloco de descrições. Como exemplo tem-se o pacote (**package**) em ADA (BOOCH, 1983, p. 185). Um pacote incorpora a descrição da estrutura de dados e das operações disponíveis sobre esta (operações exportadas ou **interface**). A implementação do pacote descreve os algoritmos utilizados para a manipulação ou mesmo simples acesso à representação. Parte deste código é de uso público, compondo a interface entre a funcionalidade do pacote e seus utilizadores.

Do ponto de vista do utilizador do pacote, apenas a interface é visível. Esta característica produz o efeito de separar os detalhes da representação dos detalhes de domínio da aplicação. Uma vez mantida a interface, uma modificação no interior de um pacote não afeta seus utilizadores. Este fato favorece a manutenção de código, uma vez que as modificações ficam restritas ao local que são efetuadas, não propagando efeitos colaterais indesejáveis para outras partes do sistema.

A figura 5.1 ilustra o pacote em ADA. Os retângulos indicam as operações exportadas. Em seu interior estão as implementações das rotinas e dados. Na representação é incluído o nome do

novo tipo de dados criado.

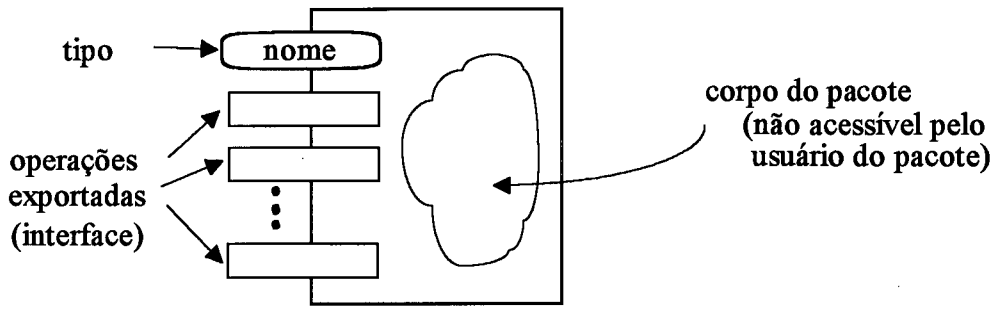


Figura 5.1 - Pacote (BOOCH, 1983)

5.7 - Classe

Na orientação por objetos a unidade básica de encapsulamento é a **classe**. A classe implementa o tipo abstrato de dados, adicionando mecanismos que caracterizam a orientação por objetos, tais como a herança e o polimorfismo. Como no pacote de ADA, estão presentes as operações disponíveis para o usuário da classe e uma estrutura de dados (figura 5.2).

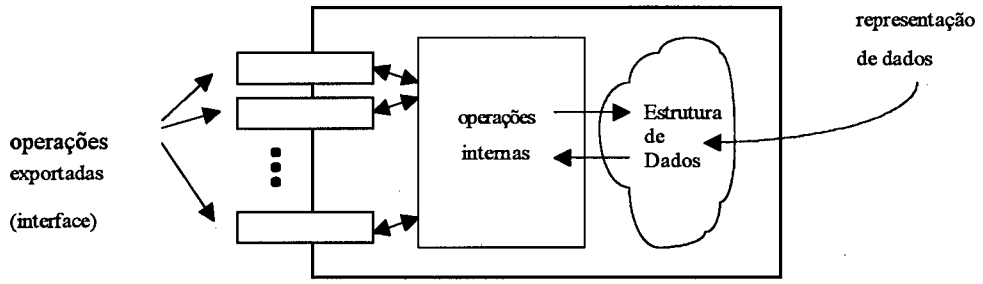


Figura 5.2 - Classe

5.8 - Passagem do problema à representação computacional

A orientação por objetos têm como idéia principal representar os elementos do problema real por abstrações no meio computacional. A classe pode ser vista como um "filtro" para esta representação. Os elementos reais são descritos na solução computacional através da observação de suas comportamentos e atributos. Cada um destes elementos é descrito por uma **classe**. Elementos com características comuns são representados por uma mesma classe (figura 5.3).

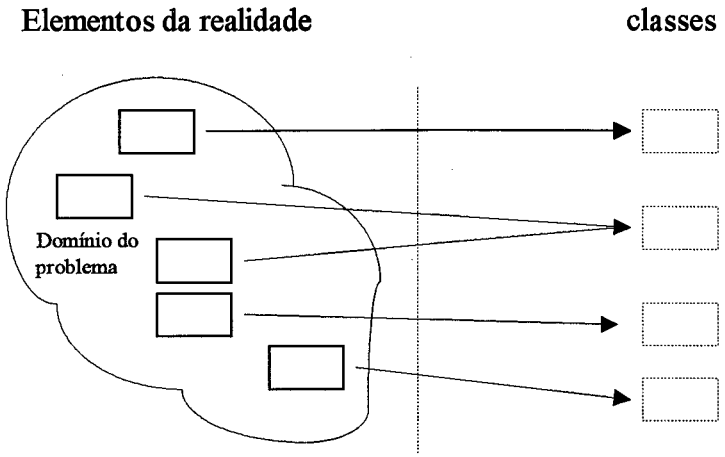


Figura 5.3 - Abstração dos elementos

Além da identificação dos elementos do **domínio da problema**, uma solução computacional também necessita da identificação de elementos do **domínio da implementação**, presentes na solução computacional, mas sem contrapartes no mundo real, tais como interfaces gráficas, listas, menus, etc.

5.9 - Criação de objetos

As classes abstraem os elementos do problema real, mas são apenas uma descrição estática de suas características. Na solução computacional são criados exemplares[5] denominados **objetos**, representando através de uma correspondência biunívoca as entidades do problema real.

No computador os objetos são espaços na memória contendo valores para os atributos da representação de dados descrita na classe que o originou. Esta classe também dá origem ao conjunto de operações que irão atuar sobre o objeto (fig. 5.4). Em um dado programa, estes objetos representam os elementos presentes no sistema modelado pelo computador.

---

[5] Neste trabalho utilizou-se a palavra **instância** para designar a criação de um exemplar de uma determinada classe, no entanto, embora muito utilizada, não é uma tradução adequada para **instance** do inglês.

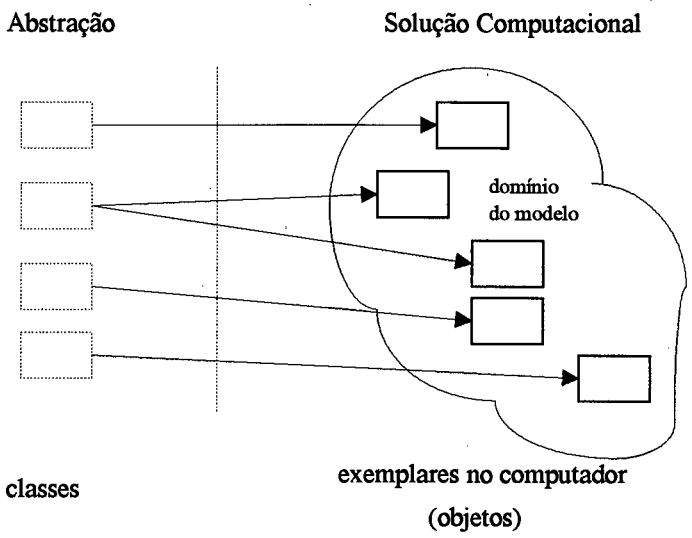


Figura 5.4 - Criação de objetos

Cada objeto possui uma identificação que denota a posição de memória que foi reservada para conter seus dados. O conjunto de métodos que implementam as operações sobre o objeto é compartilhado por todos os exemplares criados a partir da mesma classe. As operações efetuadas sobre o objeto fazem com que ele mude seu **estado atual**, o que representado pelos dados contidos no objeto.

5.10 - Mensagens

Um objeto pode interagir com outro objeto de diferentes formas. Como exemplo, um objeto pode enviar informação para outro objeto, um objeto pode requisitar informação de outro objeto, um objeto pode alterar outro objeto e ainda um objeto pode causar em um objeto a produção de uma ação (EMBLEY et al., 1992, p.167). Estas interações são produzidas através de **mensagens**, que são a ativação dos métodos definidos na interface da classe.

A figura 5.5 contém um **diagrama de fluxo de objetos** (ELLIS, 1994, p. 331) onde objetos enviam e recebem mensagens de outros elementos[6]. O símbolo >> indica a origem da mensagem.

[6] Nesta notação não necessariamente outro objeto.

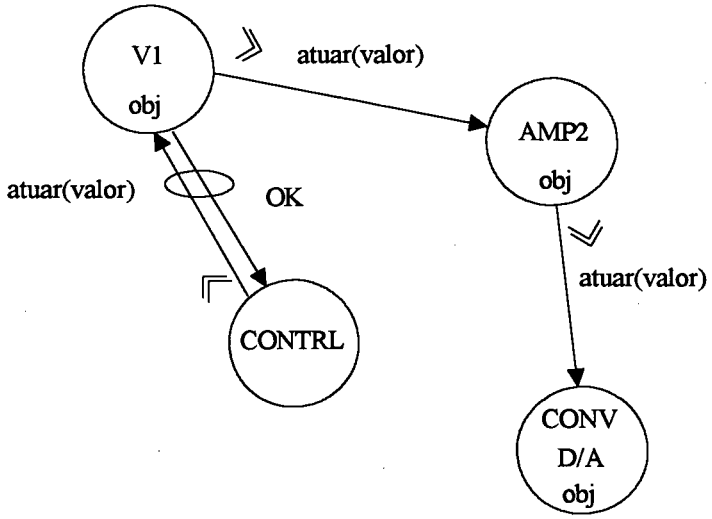


Figura 5.5 - Envio de uma mensagem para um objeto

As mensagens proporcionam o meio de transferência de informação entre os objetos e podem se originar de abstrações feitas sobre a transferência de sinais existente entre os elementos do problema. Na figura 5.5, o argumento **valor** determina uma informação sendo transferida de um objeto a objeto.

5.11 - Relações entre classes

Segundo BOOCH (1991, p.49), a possibilidade de associar classes e o mecanismo de herança é que tornam a orientação por objetos versátil. Através dos relacionamentos constrói-se ligações estáticas que fornecem indicação do fornecimento de material para a produção de novas classes (herança) e da colaboração e composição entre diferentes classes (associação e agregação).

5.11.1 - Herança

Segundo BOOCH (1994) a herança é o elemento que mais se sobressai na orientação por objetos. Através da herança é possível construir novas abstrações utilizando outras abstrações (fig.5.6). Uma nova classe pode possuir todas as características da classe utilizada como fonte, fornecendo uma especialização ou extensão desta. Isto significa que a estrutura de dados e o

conjunto de métodos da classe base se incorpora (seletivamente) à classe derivada e esta possui modificações e complementações que caracterizam seu novo comportamento.

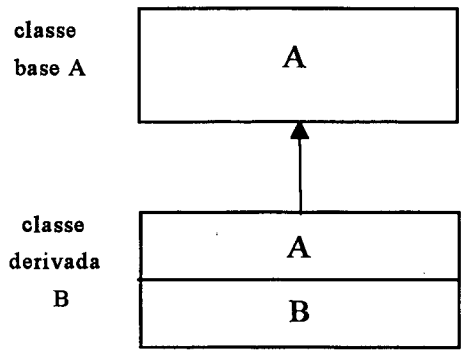


Figura 5.6 - Herança

Através de regras de visibilidade é possível controlar o que efetivamente irá ou não ser utilizado diretamente pela classe derivada. Isto permite que a nova classe modifique características da classe base, aproveitando apenas o que é necessário para a nova abstração.

Uma classe pode herdar características de uma classe (herança simples) ou de múltiplas classes (herança múltipla). O relacionamento de herança entre as classes produz uma hierarquia de classes. A classe hierarquicamente superior a uma outra classe é denominada **classe base** ou **classe superior**. A classe que herda as características de uma ou mais classes é denominada **classe derivada**. A figura 5.7 apresenta uma hierarquia de classes segundo a notação **Object Modeling Technique** (OMT / RUMBAUGH et al., 1991).

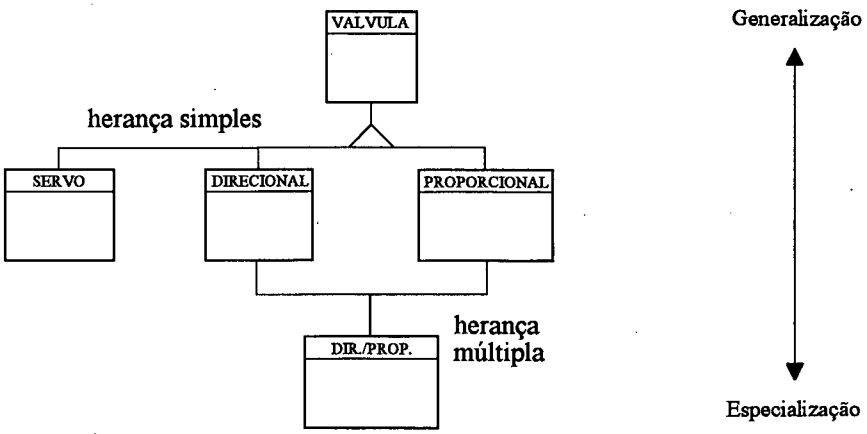


Figura 5.7 - Hierarquia de classes

Uma classe pode ser projetada apenas para fornecer características comuns a outras classes, não sendo utilizada diretamente, ou seja, não gerando objetos no programa. Este tipo de classe é denominada **classe abstrata**.

5.11.2 - Associação

Em uma associação duas classes são independentes e contêm um canal de comunicação entre elas. Uma operação sobre uma delas não afeta diretamente a outra. A figura 5.8 ilustra um exemplo também empregando a notação OMT. Neste exemplo a classe CILINDRO está associada a uma ou mais classes do tipo VALVULA. Esta associação representa, no circuito hidráulico real (físico), a possibilidade de acionamento de um cilindro por uma ou mais válvulas.

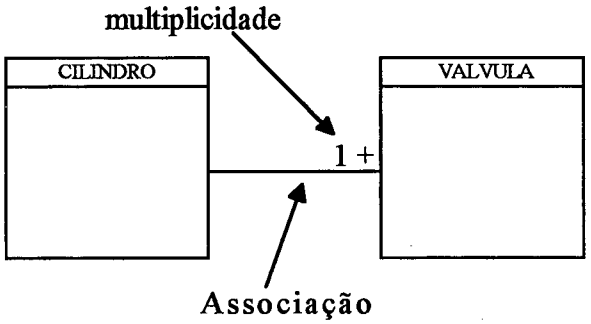


Figura 5.8 - Associação entre classes

Na execução de um programa estas associações darão lugar às ligações por onde trafegam as mensagens enviadas de objeto a objeto.

5.11.3 - Agregação

No mundo real, elementos podem ser compostos de outros elementos, como por exemplo em um sistema de atuação em vazão assistido por computador, onde uma válvula, um amplificador e um conversor DA interligados contribuem para a composição de um subsistema de atuação. Neste caso um elemento "é parte de " um outro elemento mais complexo.

A agregação captura esta situação descrevendo uma classe que é composta de outras classes. Na criação de um exemplar da classe resultante, tem-se um objeto em cujo interior outros objetos são criados (figura 5.9). Uma operação sobre o exemplar mais complexo ativa operações sobre os objetos em seu interior.

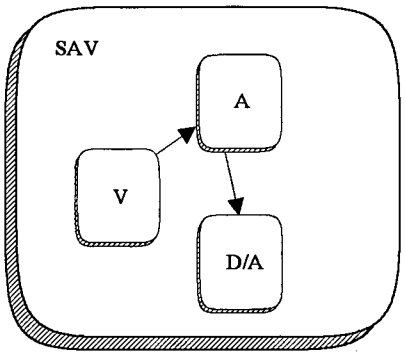


Figura 5.9 - Agregação: Um objeto contém outros objetos

Na figura 5.10 exemplifica-se a agregação de classes que abstraem um sistema de atuação em vazão como a composição de um ou mais dispositivos de diferentes tipos utilizando a notação OMT.

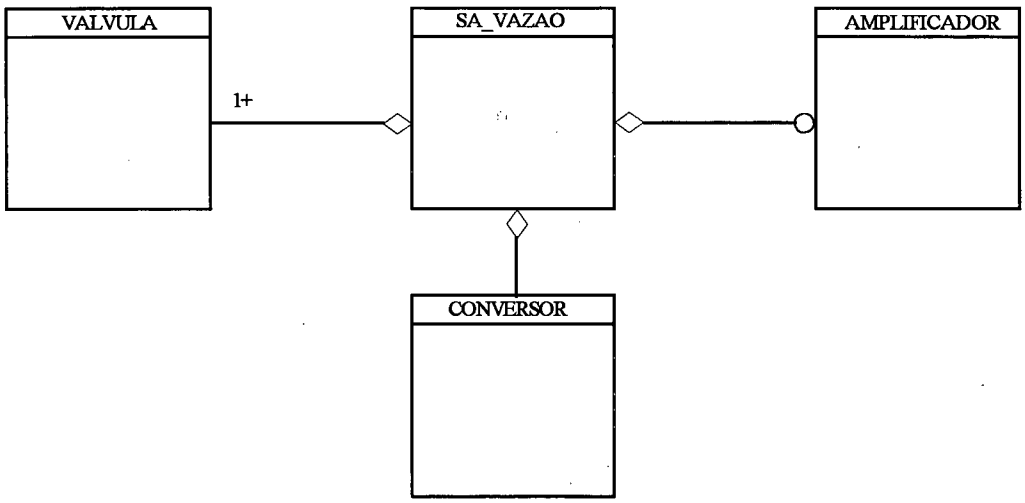


Figura 5.10- Relação de agregação entre classes utilizando OMT

Na figura, 1+ indica que uma objeto instanciado da classe S_ATUACAO pode ser constituído de um ou mais objetos do tipo VALVULA, um do tipo CONVERSOR e, de nenhum ou apenas um, do tipo AMPLIFICADOR.

Ainda segundo a notação OMT e também presente em técnicas de outros autores, como por exemplo EMBLEY et al. (1992), é possível incluir atributos às associações. Estes atributos são baseados nos diagramas de entidade-relacionamento proposto por CHEN (1976) e utilizados no desenvolvimento estruturado.

5.12 - Polimorfismo

Segundo BOOCH (1991, p. 102), **polimorfismo**[7] é um conceito na teoria dos tipos em que uma identificação (como por exemplo o nome de uma classe) pode indicar classes diferentes mas que se relacionam através de uma classe base comum (fig. 5.11).

Esta propriedade permite que o conjunto de mensagens presentes em uma aplicação não mude ao haver substituição de objetos, desde que este novo objeto seja derivado da mesma classe base do objeto anterior. Qualquer objeto que respeite a interface da classe base, ou seja, possua o mesmo conjunto de operações (ou serviços) disponíveis para o utilizador da classe, pode substituir esta, não importando sua implementação específica.

---

[7] **polimorfo**: S.m. Que se apresenta sob numerosas formas; multiforme. Dicionário Aurélio Básico da Língua Portuguesa, 1988.

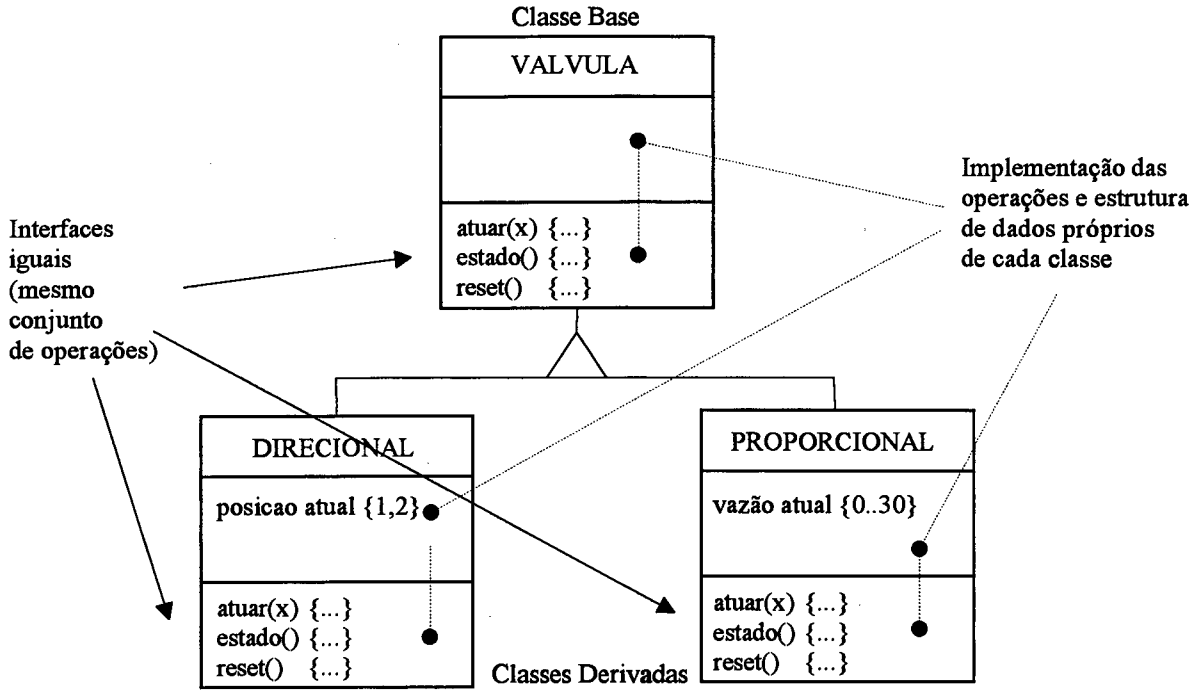


Figura 5.11 - Polimorfismo

Baseado nesta propriedade, um programa construído ao redor do polimorfismo gerência uma coleção de um tipo básico, onde se encaixam objetos derivados de uma classe comum. No caso de criação dinâmica de objetos, o resultado preciso do envio de uma mensagem para um desses objetos não pode ser determinado em tempo de compilação do programa, uma vez que o objeto presente no espaço destino da mensagem não é conhecido até o momento de sua inclusão na coleção (lista, fila, etc). Como cada classe derivada pode conter uma implementação diferente, de acordo com seu comportamento, é necessário que a mensagem destinada a um objeto criado a partir desta evoque sua implementação específica (Exemplo na fig. 5.12). A implementação correta a ser chamada deve portanto ser determinada em tempo de execução. A habilidade de retardar a resolução de implementações em tempo de execução é denominada **ligação posterior**[8].

[8] Ou ainda **ligação dinâmica**. Esta técnica se contrapõe à **ligação estática**, onde o destino da mensagem é definido em tempo de compilação do programa.

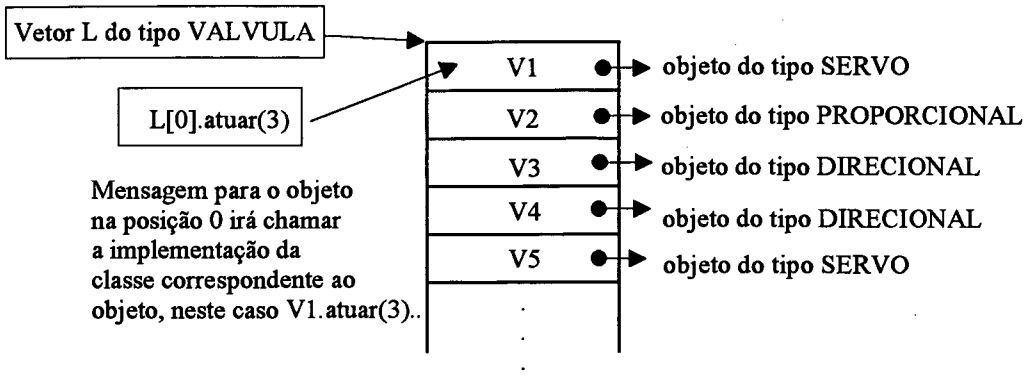


Figura 5.12 - Lista de objetos polimórficos

5.14 - Reutilização

Segundo MEYER (1990), a natureza da concepção de programas está mudando, onde o processo de industrialização de software deixa de focalizar a criação de projetos individuais e parte para a criação de componentes reutilizáveis por vários projetos. O tipo abstrato de dados implementado por classes e seus mecanismos de utilização (herança, polimorfismo) são a resposta da tecnologia atual com o propósito de facilitar a reutilização de software.

O principal fator é a natureza expansível da classe. Através da herança é possível criar novas classes a partir de outra(s) sem a necessidade de conhecer o funcionamento dos métodos já implementados. Seguindo este raciocínio, a construção de um novo sistema computacional (**software**) pode ser iniciado tendo por base um conjunto de classes já definidas, adaptáveis por natureza às novas necessidades. Outro fator importante é a possibilidade de incluir novas classes sem que a aplicação sofra alterações (polimorfismo). Neste contexto, um programa que manipule uma determinada válvula hidráulica é capaz de manipular qualquer outro tipo de válvula com mesma interface, o que contribui para simplificar manutenções e evoluções no sistema.

5.15 - Engenharia de Software - Ciclo de Vida

Não existe ainda um modelo de ciclo-de-vida de aceitação geral para o desenvolvimento de software voltado para a orientação por objetos. Normalmente o modelo utilizado é baseado no ciclo de vida convencional (**water fall**), modificando o enfoque habitualmente funcional, direcionando-o para a dualidade objeto/classe e seus relacionamentos. Em adição, há também a necessidade da manipulação de bibliotecas de classes, na forma de utilização do que já foi desenvolvido e inclusão de novos elementos, tanto na fase de projeto como na fase de codificação. Todo o processo do ciclo de vida se baseia na criação de modelos estáticos e dinâmicos, utilizando-se na maioria das vezes ferramentas clássicas, como diagramas de fluxos de dados e máquinas de estado. Segundo NERSON (1992), comparada a técnicas convencionais, o desenvolvimento orientado por objetos é um processo de transformação linear, não existindo uma mudança de paradigma nos diferentes estágios do ciclo-de-vida. Se tipos de objetos são identificados durante a etapa de análise e especificados por um nome e um conjunto preciso de propriedades, eles são traduzidos em unidades sintáticas no programa final (classes).

Um ponto em comum entre vários autores é a forma incremental de desenvolvimento do processo, com sobreposições entre as fases. O desenvolvimento orientado por objetos permite e incentiva a criação de modelos incompletos em cada etapa, que vão se enriquecendo até a completude adequada. Este procedimento incremental tem o respaldo inclusive no momento de codificação, já que classes podem ser elaboradas do ponto de vista de suas interfaces, onde é possível uma verificação sintática ainda na fase de definição do comportamento das classes.

A figura 5.13 esquematiza o tempos e graus de atividade associados ao método proposto por SELLERS & EDWARDS (1990, p158), bem representativo do processo orientado por objetos.

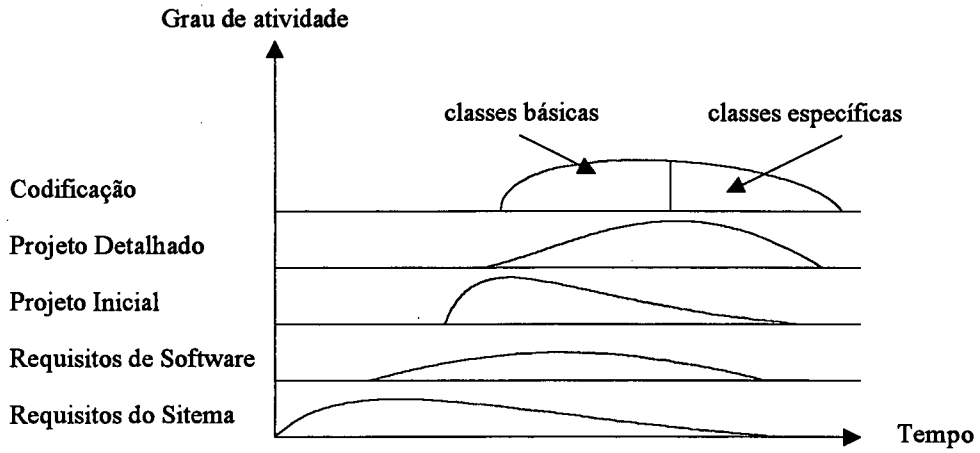


Figura 5.13 - Grau de atividade como função do tempo nos vários estágios no desenvolvimento orientado por objetos (HENDERSON-SELLERS & EDWARDS, 1990).

Outras metodologias enfatizam a construção de um modelo inicial operacional do sistema e a adição de novos atributos até a versão final (figura 5.14). Este processo é denominado **desenvolvimento em espiral** (BOEHM,1986. citado por JACOBSEN et at. 1993). O modelo em espiral descreve como um produto de desenvolve para formar novas versões e como uma versão sofre incrementos do protótipo até um produto completo.

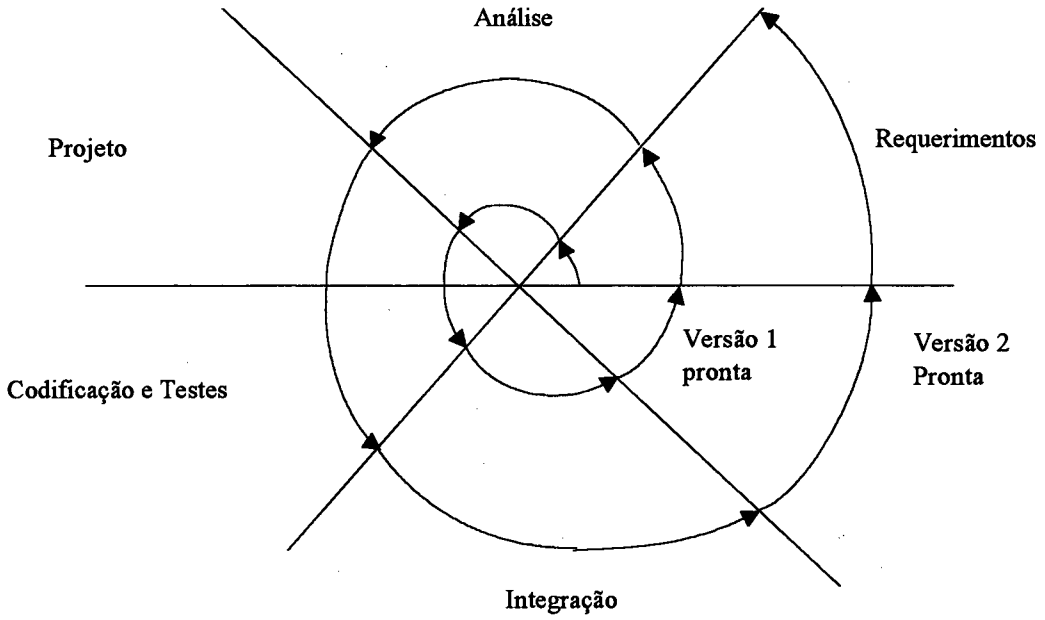


Figura 5.14 - Desenvolvimento em espiral (JACOBSEN et al., 1993)

Nem todos os métodos utilizam o paradigma em todas as fases. Métodos, como OOD (BOOCH, 1991) e HOOD (1993), utilizam o produto da análise funcional como base para o projeto orientado por objetos. De um modo geral, autores como BOOCH (1991), SELLERS & EDWARDS (1990), RUMBAUGH et al. (1991), JACOBSEN et al. (1992), são permissíveis quanto a utilização de linguagens não orientadas por objeto na fase de implementação[9].

A figura 5.15 ilustra uma rede C/A com uma generalização das etapas e os modelos produzidos no desenvolvimento orientado por objetos.

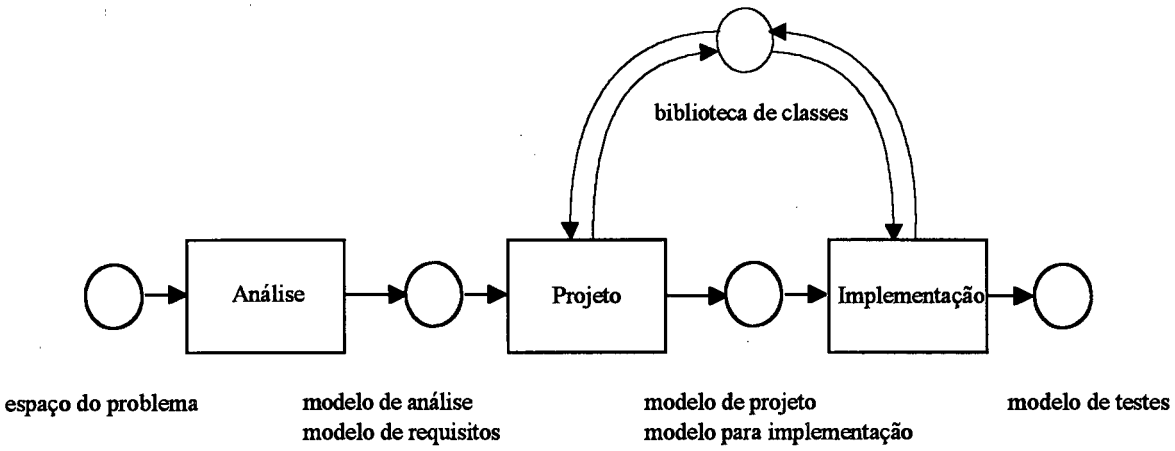


Figura 5.15 - Desenvolvimento na engenharia de software orientado por objetos

Cada modelo produzido nas diferentes fases geralmente possui, além da documentação textual[10], uma forma estática de representação do sistema, a **hierarquia de classes;** e duas formas dinâmicas: um **diagrama funcional,** que descreve a estrutura do sistema e passagens de informação; e um **diagrama comportamental,** que descreve a evolução do sistema em função do tempo.

5.15.1 - Hierarquia de classes (descrição estática)

Produz modelos estáticos do sistema, compondo-se de classes e seus relacionamentos (especialização, generalização, agregação e associação). Para sua representação são utilizados, de

---

[9] Neste caso a implementação orientada por objetos seria vista como um estilo de programação.

[10] Dicionários, descrição em linguagem corrente, etc.

uma forma geral, variantes dos diagramas de entidade/relacionamento (ER), incluindo ligações para indicar o mecanismo de herança, como nos métodos OOA/COAD & YOURDON (1991) e OMT/ RUMBAUGH et al. (1991), OSA/EMBLEY(1992).

Após a identificação (etapa de análise) dos elementos do espaço do problema e seus relacionamentos, estes são classificados e documentados. Nas etapas subseqüentes este modelo sofre detalhamentos, outras abstrações são acrescentadas e algumas suprimidas, mudanças que ocorrem em conjunto com modificações nos relacionamentos entre as classes.

5.15.2 - Diagrama funcional (descrição dinâmica)

O modelo funcional se preocupa com as transformações sofridas pela informação que circula pelo sistema. Para construir o modelo funcional no desenvolvimento orientado por objetos é necessário a inclusão de uma representação para as mensagens enviadas e recebidas pelos objetos. Alguns autores utilizam variantes de métodos desenvolvidos na análise funcional, notadamente os diagramas de fluxo de dados (DFD), outros propõem uma representação própria. Métodos como OMT utilizam irrestritamente o DFD. SELLERS & EDWARDS (1990) advogam a utilização do **diagrama de fluxo de informação** (IFD), análogo ao diagrama de fluxo de dados (DFD), onde o termo **informação** é resultante do fato de os dados não fluírem no sistema da mesma forma utilizada em um sistema funcional descrito por um DFD, mas sim estarem contidos no interior dos objetos. Não são por conseguinte representados os depósitos de dados.

Já autores como BOOCH (1991), JACOBSEN (1992) utilizam notações próprias. O método OOD / BOOCH (1991) inclui em seus diagrama a possibilidade de representar ligações entre **objetos ativos**[11]. ELLIS (1994) descreve o **diagrama de fluxo de objetos**, possuindo notação para representar objetos, funções (processadores de dados), elementos de controle, depósitos de dados, origem de mensagens, tipos de fluxos (dados, eventos, mensagens), incluindo o sentido destes. Segundo ELLIS (1994, p.331) a inclusão de funções e depósitos de dados no modelo

---

[11] Definido no capítulo 5

orientado por objetos é permitida, uma vez que de um ponto de vista prático, não existe uma obrigatoriedade em transformar todos os elementos de um sistema em objetos, havendo momentos em que faz sentido mesclar entidades não descritas por classes à solução.

5.15.3 - Modelo comportamental (descrição dinâmica)

O modelo comportamental abstrai os estados dos elementos reais, que evoluem em função de eventos e ações presentes no sistema modelado, representando a noção de temporalidade do sistema. Os métodos de desenvolvimento utilizam fundamentalmente máquinas de estado, diagramas de interações, fluxo de eventos ou diagramas de tempo. Segundo RUMBAUGH et al. (1992) o modelo dinâmico consiste em múltiplos diagramas de estado, existindo um diagrama de estados para cada classe com comportamento significante. Cada máquina de estado do sistema tem execução concorrente e combinam-se em um modelo dinâmico unificado, representando o comportamento de todo o sistema através de eventos compartilhados. No decorrer do tempo, os objetos estimulam uns aos outros, resultando em uma série de mudanças em seus estados. Eventos e ações têm seus mecanismos implementados através da ativação dos métodos de suas interfaces.

6 - A Codificação

6.1 - Introdução

A codificação de um projeto orientado por objetos é substancialmente facilitada com o uso de uma linguagem orientada por objetos, uma vez que as construções baseadas no paradigma são mapeadas diretamente em elementos da linguagem. Por outro lado, a utilização de uma linguagem não orientada por objetos na codificação de um projeto orientado por objetos, embora possível, requer grande cuidado e disciplina na preservação da estrutura orientada por objetos do projeto, e o programador não pode obter ajuda na linguagem caso violações sejam produzidas.

Este capítulo aborda a utilização da linguagem C++, indicada para o uso em automação pela sua característica híbrida, onde é possível a chamada de funções independentes (não pertencentes a nenhuma classe).

6.2 - Fase de codificação

Nas primeiras etapas do desenvolvimento orientado por objetos (análise e projeto), os objetos encontrados no domínio do problema são diretamente mapeados em objetos no domínio da solução. Se o mesmo paradigma é empregado no ambiente de codificação, obtém-se um segundo mapeamento direto. Segundo BAILIN (1989), o uso de um mesmo paradigma em todo o desenvolvimento é visto como uma das grandes virtudes da orientação por objetos. Muitas das metodologias em desenvolvimento atualmente dão grande importância a codificação, uma vez que a análise sendo um processo visto como **top-down**, pode ser complementado pela codificação **bottom-up**, onde são identificadas classes existentes em bibliotecas, estendidas pela herança e construção de novas classes quando adequado.

6.3 - Reusabilidade

Outro ponto favorável à utilização de linguagens orientadas por objeto é a reusabilidade alcançada, o que reduz esforços com projeto, codificação e testes. A reutilização está presente em linguagens não orientadas por objetos, mas é um fator otimizado nas orientadas.

Segundo RUMBAUGH et al. (1991) existem duas formas clássicas de reutilização na fase de codificação da engenharia de software:

1. O compartilhamento de código recém escrito dentro de um projeto ou reutilização de código previamente escrito em um novo projeto. Isto geralmente implica na descoberta de seqüências de código redundante e uso de recursos da linguagem, como funções ou macros. O beneficio imediato é a redução do tamanho dos programas e facilidade na verificação de código (**debug**);

2. A utilização de código previamente desenvolvido. Este tipo de uso requer um planejamento maior e representa um investimento. Geralmente são utilizadas bibliotecas relativamente específicas, tais como pacotes gráficos e bibliotecas de análise numérica .

Em linguagens convencionais a reutilização é construída ao redor de coleções de funções, as quais geralmente são bem específicas, não existindo a possibilidade de reaproveitá-las para uma estrutura de dados diferente daquela para qual foram projetadas .

Em uma linguagem orientada por objetos a reutilização está construída ao redor de **classes** e da **herança**. Um conjunto de classes fornece a solução para estruturas de dados básicas, que podem ser alteradas e estendidas em função da especialização planejada. Além da reutilização do código como fornecido, ele pode ser adaptado para novas situações, mesmo não previstas por

seus idealizadores. Graças ao mecanismo de herança não é necessário possuir o código fonte para realizar adaptações, fato que diferencia as linguagens orientadas por objetos de outras linguagens.

6.4 - Custos

Existe um custo de desempenho no envio de uma mensagem de um objeto a outro em uma linguagem orientada por objetos. Para as chamadas de métodos que não podem ser resolvidas de modo estático (momento da compilação), a codificação deve efetuar uma procura dinâmica para localizar o método definido para a classe do objeto receptor. Estudos indicam (PASCOE, 1986, p. 144) que a chamada dinâmica de um método toma de 1,75 a 2,5 vezes o tempo gasto na chamada de uma sub-rotina simples. No entanto a experiência mostra que a resolução dinâmica é necessária em apenas **20%** do total de chamadas em um programa. Além disto, nas linguagens "fortemente tipadas"[12], o compilador geralmente consegue determinar quais chamadas podem ser resolvidas de forma estática e gerar código para a chamada de uma sub-rotina em lugar de algum sistema de procura.

Outra fonte de ineficiência não provém da própria linguagem, mas de sua utilização associada ao desenvolvimento orientado por objetos que leva à construção de sistemas com vários níveis de abstração. Desta forma, métodos de classes inferiores são utilizados apenas para a chamada de métodos de classes superiores mais complexos ou para ganho de acesso a campos protegidos. A chamada de um método em um nível superior de abstração usualmente resulta em uma cascata de chamadas aos outros métodos em níveis inferiores. Para aplicações em que o tempo é um recurso limitado, este efeito pode ser inaceitável, no entanto, estes níveis de abstrações são essenciais para a inteligibilidade do sistema. (BOOCH, 1991, p.216).

Uma solução para o problema da perda de eficiência é construir um programa visando sua funcionalidade e posteriormente executá-lo e determinar os pontos em que são necessárias

---

[12] Característica de linguagens que garantem em todas as expressões a consistência em relação aos tipos empregados.

otimizações em função do tempo de execução (ELLIS, 1994, p. 222). Alguns artifícios para aliviar sobrecargas impostas com custos de chamadas podem ser utilizados, tais como transformar chamadas de sub-rotinas problemáticas em expansões de código (funções **inline**), desencapsular certos campos ou, em casos extremos, transformar objetos em simples estruturas de dados.

Quanto ao uso de linguagens de alto nível em sistemas automáticos, é comum o conceito errôneo, como aponta SELIC et al. (1994), da necessidade de se utilizar linguagens assembler para a obtenção de tempos de resposta na ordem de microsegundos. Segundo este autor, tempos de acesso desta grandeza se aplicam apenas a uma categoria de sistemas.

6.5 - Concorrência - Objetos Ativos

Segundo autores como SELEC et al. (1994) e ELLIS (1994), uma das características dos **sistemas de aquisição e atuação**[13] é a necessidade da utilização da concorrência.

No paradigma orientado por objetos com estrutura seqüencial, a chamada de uma operação sobre um objeto é um processo **síncrono**, onde os objetos são entidades passivas, mudando de estado apenas quando seus métodos são evocados (KARAORMAN & BRUNO, 1993, p. 104). Concorrência significa a possibilidade de execução paralela, permitindo intercalar os processamentos de componentes de um programa em um único processador ou distribuído por vários processadores. Segundo MINOURA et al. (1993), os objetos passivos utilizados na programação orientada por objetos convencional não permitem um encapsulamento e modularização de controle adequados quando são utilizados para modelar objetos do mundo real.

Em um nível alto de abstração, a programação orientada por objetos pode aliviar o problema da concorrência na maioria dos programas, escondendo seu mecanismo no interior de abstrações reutilizáveis. Os modelos de computação concorrentes baseados em objetos especificam como os estes interagem e como as mensagens são passadas e recebidas. Um objeto

---

[13] Identificados por estes autores como **sistemas em tempo real.**

ativo é a integração da noção de processo com a noção de objeto. Desta forma, um objeto ativo é simplesmente um objeto que contém sua própria seqüência de controle de execução. Objetos ativos são geralmente autônomos, ou seja, podem exibir algum comportamento sem que algum de seus métodos estejam sendo solicitados por outro objeto (BOOCH, 1991, p.83).

Pode-se identificar 3 formas diferentes de se introduzir a concorrência na programação orientada por objetos:

1. Projetar uma nova linguagem;

2. Estender um linguagem orientada por objetos já existente ou;

3. Desenvolver um biblioteca e incorporá-la a uma linguagem orientada por objeto já existente.

A maioria dos projetistas utilizam a terceira possibilidade por razões de simplicidade e flexibilidade.

6.6 - C++

C++ foi desenvolvida por Bjarne Stroustrup nos laboratórios Bell da empresa AT&T no início da década de 80, utilizada inicialmente para a codificação em sistemas de chaveamento e em uma versão do sistema operacional UNIX (ECKEL, 1989, p20). Um dos propósitos de sua construção foi permanecer compatível e comparável à linguagem C em termos de sintaxe, desempenho e portabilidade (JORDAN, 1990, p.61).

Além de implementar o paradigma, C++ é capaz de utilizar os recursos da programação procedural, tornando-se uma linguagem de alto nível candidata natural para a codificação de um projeto orientada por objetos para sistemas técnicos. C++ é uma linguagem "tipada", ou seja, as operações possíveis para uma classe são específicas para esta, permitindo que conflitos sejam

resolvidos em tempo de compilação e erros sejam indicados cedo. Isto a diferencia de Smalltalk e linguagens similares no sentido em que estas fornecem mais um "ambiente" de programação. Segundo ECKEL (1989), muitas implementações de Smalltalk chegam a substituir o sistema operacional, onde a resolução de todas as funções são feitas em tempo de execução e não existe verificação de tipos estática (a linguagem não possui "tipagem"), onde todos os objetos são derivados de uma mesma classe base. Estes fatores trazem um custo adicional de execução que prejudica sistemas que requerem E/S

Até pouco tempo C++ era discriminada em relação a falta de alguns recursos encontrados em outras linguagens de alto nível, como por exemplo **garbage collection** e **exception handling**. No entanto tais propriedades podem ser programadas em C++ sem que e a linguagem seja estendida (CAMPBELL et al., 1993).

No caso específico de **garbage collection**[14], ELLIS (1994, p.160) argumenta não ser este um atributo favorável na construção de programas para uso em tempo real, uma vez que não é previsível o início e duração de tal atividade, sendo portanto, um recurso de manipulação cuidadosa. Compiladores mais modernos possuem gerenciadores de memória que satisfazem em grande parte esta necessidade e também já se encontra presente o recurso de **exception handling**[15] (Ex.: WATCOM C/C++) .

Outro fator positivo de C++ é a compatibilidade de ligação com rotinas de outras linguagens como FORTRAN, Pascal, Assembler e o próprio C, o que admite uma incorporação imediata de uma enorme quantidade de código já produzido e testado, o que se traduz em um auto grau de reutilização.

---

[14] Manutenção automática de memória liberada a fim de evitar fragmentações e erros por falta de memória.

[15] Evocação automática de uma rotina para tratar uma condição de erro que transforma a seqüência de execução de um código imprática ou ilógica, operando como um processo em separado.

6.7 - Definição de novos tipos

C++ implementa o tipo abstrato de dados através da classe, que de uma forma prática cria um tipo de dado definido pelo programador, de manipulação semelhante a dos tipos pré-definidos na linguagem. Uma classe possui uma estrutura de dados composta por exemplares de variáveis básicas fornecidas pela linguagem (int, float, char, etc.) e/ou por outras classes, mais um conjunto de rotinas.

A definição do que é encapsulado e o que faz parte da interface com o usuário é determinado por regras de acesso indicado pelo programador através de três níveis de controle, sinalizados pelas palavras-chave (figura 6.1) :

private - Indica os elementos encapsulados, os quais só podem ser acessados por outros elementos pertencentes à classe;

protected - Elementos encapsulados mas que são de acesso livre para classes diretamente derivadas. Este procedimento causa um relaxamento no encapsulamento[16];

public - Indica os métodos que estão disponíveis para chamadas externas. Caso algum dado seja declarado como public, também estará livre para manipulações externas.

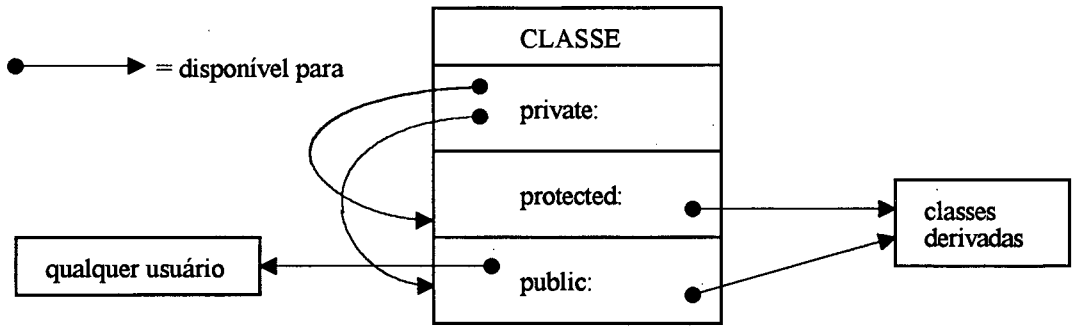


Figura 6. 1 - Controle de acesso

Utilizando-se uma válvula hidráulica como base para a construção de uma abstração, tem-se uma possível codificação em C++ no código seguinte:

---

[16] Outra forma de quebra de encapsulamento é a utilização de funções ou outras classes declaradas como "amigas" (**friend**), que obtém acesso aos membros **private** ou **protected** de outra classe. Estes recursos violam o encapsulamento, devendo ser usados com parcimônia a fim de não comprometer a independência de código.

```
class VALVULA {
private:
        char *Id;      // guarda a identificação da valvula
protected:
        FT   *pft; // ponteiro para função de transferencia
        AMPLIFICADOR *amp; // ponteiro para aplificador
public:
        VALVULA(char *ID, AMPLIFICADOR *amp);
        virtual void atuar(double QV)=0;  // vazão desejada
        virtual double estado(); // retorna o estado  corrente da valvula
};
```

Esta classe não descreve um tipo determinado de válvula, mas uma base genérica para a construção de classes derivadas mais específicas. Como esta classe não deve produzir instâncias no programa ela é denominada **abstrata**, o que é indicado pela falta de corpo de um ou mais métodos (igualados a zero na declaração da classe). Como exemplo, pode-se construir uma hierarquia (Fig. 6.2) composta de dois tipos de válvulas derivadas, V_SERVO e V_PROPORCIONAL, descrevendo respectivamente uma servoválvula e uma válvula proporcional.

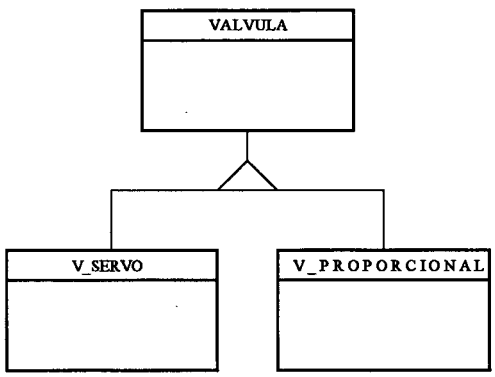


Figura 6.2 - Hierarquia para o exemplo da válvula

O código para descrição das classes conteria:

```
class V_SERVO:public VALVULA  {
private:

        double qvd_atual;        // vazao desejada atual
        double qvmax;            // vazao maxima possivel
        double  idx;        // corrente de pilotagem
public:
        V_SERVO(char *ID, double QVMAX, AMPLIFICADOR *AMP,
                        PTF P):VALVULA(ID,P,AMP); // construtor
        ~V_SERVO();
        void atuar(double QV); //
```

```
        double estado();  // retorna o estado  corrente da valvula
};

class V_PROPORCIONAL:public VALVULA {
private:
        double qvd_atual;       // vazão desejada atual
        double qvmax;           // vazão máxima possível
        double idx;         // corrente para pilotagem eixo x
        double idy;         // corrente para pilotagem eixo y
        AMPLIFICADOR *amp2; // ligação com um amplificador
                            // adicional
public:
        V_PROPORCIONAL(char *ID, double QVMAX, AMPLIFICADOR *AMP1,
              AMPLIFICADOR *AMP2, PTF p):VALVULA(ID,P,AMP1); //construtor
        ~V_PROPORCIONAL();
        void atuar(double QV);
        double estado();  // retorna o estado  corrente da válvula
};
```

O método construtor é chamado apenas no momento da criação de um objeto (instanciação), recebendo parâmetros necessários para a inicialização e posicionamento deste em um estado inicial. Na seguinte linha de código é exemplificada a criação do objeto v1 do tipo V_SERVO:

```
V_SERVO v1( "v1", 25e-5, &a1, &ft1 );
```

Os seguintes parâmetros são utilizados:

- **"v1"** um nome para o objeto, semelhante à identificação de componente em um esquema elétrico, hidráulico, etc.;
- **25e-5** o valor máximo para a vazão da válvula, ou seja, vazão máxima que este exemplar de válvula poderá operar;
- **&a1** endereço de um objeto amplificador de potência. Aqui é exemplificada a codificação de uma associação entre dois objetos, estabelecendo um caminho de comunicação entre estes;

- **&ft1** endereço de uma função de transferência para esta válvula (neste trabalho a função de transferência também constitui um objeto.

Esta é uma inicialização possível entre qualquer outra escolhida pelo projetista, podendo a classe possuir vários construtores. Os valores iniciais não são obrigatórios, sendo possível criar uma função construtora que não forneça nenhum valor inicial. Parte dos valores fornecidos são transferidos para a classe base, o que também é descrito na função construtora, evocando automaticamente a construtora da classe base. A ordem de construção é iniciada pela classe superior, passando pelas classes intermediárias até a classe do objeto sendo criado. Como pode ser observado, a classe base é reutilizada para a construção de classes mais específicas, neste caso expandindo sua estrutura de dados e funcionalidade.

C++ não possui nenhum mecanismo automático de liberação de memória para objetos que não são mais referenciados no programa (**garbage collection**), mas permite que estes liberem a memória requisitada ao saírem de escopo. Embora isto requeira um cuidado a mais pelo programador, elimina o gasto de tempo envolvido com a limpeza automática. A função com o nome da classe precedida de um til (~) indica a **função destruidora**, que é chamada automaticamente assim que o objeto sai de escopo, cumprindo a função de liberação de memória. Caso a construção da classe requeira alocação dinâmica de memória para alguns de seus elementos internos, é necessário que a função destruidora contenha código para sua liberação.

A classe base VALVULA não possui muitos elementos a serem reutilizados, mas cumpre um papel importante na hierarquia: através dela é construído o **polimorfismo**. Para que um programa manipule objetos derivados de VALVULA da mesma maneira, a classe base fornece a interface comum. Cada objeto derivado possui a interface com as mesmas funções. Um ponteiro do tipo VALVULA pode apontar para qualquer objeto derivado e através dele serem enviadas mensagens que disparam os métodos específicos para cada objeto. Isto permite que novos tipos

derivados sejam introduzidos em um sistema sem que seja alterada a forma com que o programa se comunica com estes, ou seja, o programa "sabe" como se comunicar com os objetos derivados, que por sua vez, "sabem" o que fazer com as mensagens recebidas.

```
void main()
{
        AMPLIFICADOR  a1, a2, a3;          // amplificadores
        FT_PROPORCIONAL ft_prop; // função de transf. para v. proporc.
        FT_SERVO ft_serv;          // função de transf. para servoval.
        VALVULA *ptr_valv;         // este ponteiro pode apontar para
                                   // qualquer objeto derivado de VALVULA
        // criação de um objeto vserv1
        V_SERVO vserv1( "v1", 25e-5, &a1, &ft_serv );
        ptr_valv = &vserv1;      // ptr aponta para vserv1

        vserv1.atuar(5e-5);        // a vávula vserv1 recebe como vazão
                                   // desejada 5x10-5 l/m
        ptr->atuar(6e-5);  // a vávula vserv1 recebe como vazão
                           // desejada 6x10-5 l/m
        // criação de um objeto vprop1
        V_PROPORCIONAL vprop1( "v2 prop", 33.33e-5, &a2, &a3,&ft_prop );
        vprop1.atuar(9e-5);        // a vávula vprop1 recebe como vazão
                                   // desejada 9x10-5 l/m
        ptr_valv = &vprop1;      // ptr_valv agora aponta para valprop1
        ptr_valv.atuar(0);       // a vávula vprop1 recebe como vazão
                                   // desejada 0 l/m
};
```

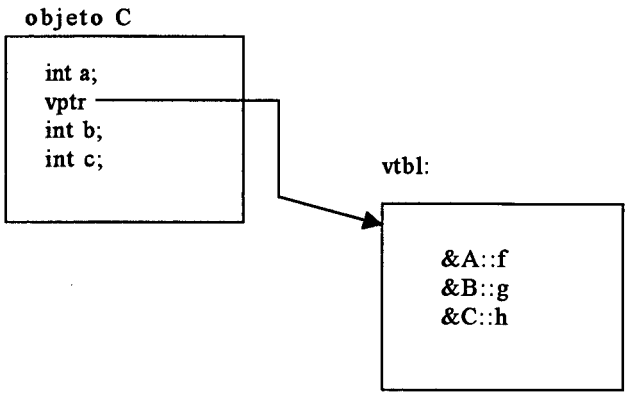
Neste exemplo de código foi utilizada a herança simples. C++ também suporta herança múltipla, com um mecanismo de exclusão automática para o caso de ocorrência de herança indireta da mesma classe base mais de uma vez (uso da palavra-chave **virtual** antes na declaração da classe base).

6.8 - Polimorfismo

Em C++ não existe um processo de procura do método adequado ao objeto que recebe uma mensagem, como acontece em Smalltalk, mas um sistema de resolução de ponteiros. Para cada função declarada como **virtual** na classe base, é fornecido um ponteiro que aponta para a função correta do objeto receptor no momento de uma chamada. No exemplo a seguir é ilustrado o

processo implementado através de tabelas de funções, denominadas VTABLE, criadas para cada objeto (ELLIS & STROUSTRUP, 1993):

```
class A
{
public:
        int a;
        virtual void f();
        virtual void g();
        virtual void h();
}
class B: public A
{
public:
        int b;
        void g();
};
class C : public B
{
public:
        int c;
        void h();
}
```



Um objeto do tipo C possui elementos herdados das classes A (int a), B (int b) e dela própria (int c). A classe base A declara como virtuais os métodos f, g e h. Isto leva a construção de uma tabela de resolução de métodos para cada objeto das classes derivadas. Esta tabela indica o método correto para o objeto, ou seja, ao ser chamado um método, o escopo é determinado pela tabela. Em um programa geralmente são manipulados objetos com apontadores do tipo da classe base. Qualquer adição de objetos de classes derivadas permite que esta manipulação ainda seja válida. Isto significa que o objeto receptor é indeterminado em tempo de execução do programa, pois o ponteiro pode apontar para qualquer objeto derivado de uma classe base.

6.9 - Eficiência em C++

Linguagens de alto nível sempre foram preteridas às montadoras na codificação de sistemas em tempo real, uma vez que aquelas podem extrair o máximo de eficiência da arquitetura da máquina em que o programa é executado. Atualmente este cenário se modificou, sendo necessário um ótimo programador em assembler para produzir um código tão eficiente quanto os produzidos por otimizações alcançadas com os compiladores atuais. De qualquer forma, sempre é possível introduzir chamadas a funções escritas em assembler de dentro dos programas. Isto ocorre em linguagens como Pascal, C ou FORTRAN, sendo o método utilizado para extrair o melhor dos dois mundos.

Além de serem de alto nível, as linguagens orientadas por objetos têm a fama de serem ineficientes em decorrência do funcionamento interpretado das linguagens mais antigas, como Smalltalk e Lisp (RUMBAUGH, 1991, p.319). O desenvolvimento de compiladores e de interpretadores mais eficientes mudou este quadro.

Quando foi criada a linguagem C++ suas construções eram traduzidas para C e depois compiladas, se beneficiando da grande portabilidade desta técnica, porém com sacrifícios em relação a depuração. Atualmente os compiladores produzem o código executável diretamente, o que aumentou consideravelmente seu desempenho. A linguagem C é considerada eficiente em função do tempo de execução do código gerado. Como a linguagem é procedural, o tempo envolvido na chamada de uma função é o parâmetro para a medição de eficiência, que pode ser adotado como referência de comparação (FIEDLER et al., 1991, p.182).

Em C++ existe um custo adicional na chamada de um método de uma classe em relação a uma função em C. Outro fator de importância é o problema de resolução de chamadas efetuado em tempo de execução, onde existe um ponteiro adicional armazenado para a tabela VTABLE, construída para cada objeto derivado de uma classe que contenha funções virtuais.

Neste trabalho foram utilizados dois métodos para verificação do custo em tempo nas chamadas. O primeiro método utilizou o software PROFILE da MICROSOFT, utilizado para a descoberta de pontos que requerem otimização em programas, que foi no entanto descartado, pois em cada execução do experimento os resultados não se repetiam. A segunda opção foi a utilização de um recurso de hardware relativamente simples, mas que demonstrou uma repetibilidade constante. A figura 6.3 ilustra o arranjo utilizado, onde um osciloscópio é conectado a uma linha de saída de uma porta paralela do computador IBM-PC monitorando um trem de pulsos gerado pelo programa de teste .

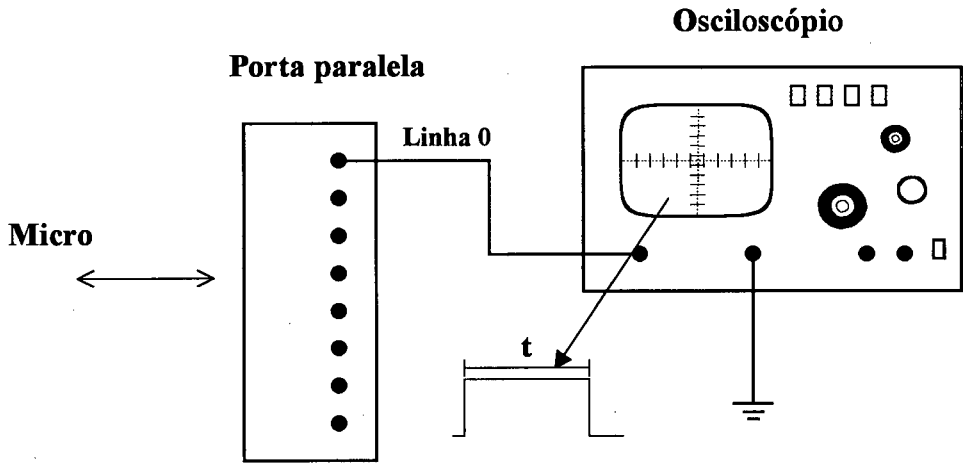


Figura 6.3 - Arranjo utilizado para a medição de tempo de chamada de uma função ou método.

Antes de ser executada uma chamada a uma função ou método, o bit da porta é ligado (nível de sinal alto), a chamada é executada e logo após o bit é desligado. O tempo (t) entre o início da subida do sinal e o fim de sua descida é então utilizado para comparações. Para que haja uma maior precisão, o tempo de um ciclo vazio foi medido e subtraído em cada processo completo de chamada. Devido a resolução do processo e instrumental utilizado, foram acrescentados dois laços: o primeiro permitindo a geração de vários ciclos para que o sinal possa ser retido pelo osciloscópio; o segundo aumentando o espaço entre subida e descida do sinal,

chamando várias vezes a função ou método para que, devido a resolução horizontal do osciloscópio utilizado, pudesse ser feita a avaliação.

```
for(c=0;c<ciclo;c++)
{
  OUT_HI
    for(t = 0; t < n_chamadas; t++)
    outG();
  OUT_LOW
}
```

O resultado obtido tomou em consideração o tempo envolvido na chamada de uma função global, estabelecida como parâmetro para comparação com as outras chamadas.

O programa analisado foi gerado através do compilador MICROSOFT C7, sem otimizações e com otimizações. No código fonte foram especificados pontos em que eram efetuadas chamadas para funções em 6 condições:

1. **Funções com escopo global** - forma utilizadas como referência, pois não pertencem a nenhuma classe, apresentando o comportamento equivalente em C;

2. **Funções pertencentes a uma classe base sem elementos virtuais**. São funções que implementam os métodos das classes, e são chamadas especificando-se o objeto pertencente. Ex.: `Objeto.metodo();`

3. **Funções pertencentes a uma classe derivada**. No exemplo, a classe derivada herda as funções da classe base. Desta forma, as funções chamadas pertencem ao escopo da classe base;

4. **Funções pertencentes a uma classe base que contém funções virtuais**. São equivalentes às pertencentes a uma classe base não virtual;

5. **Funções pertencentes a uma classe derivada de uma classe base virtual.** Neste caso possuem a VTABLE. Neste exemplo, as funções foram implementadas para a classe derivada, sendo estas as efetivamente chamadas;

6. **Funções com resolução em tempo de execução (late binding).** São métodos resolvidos através de um ponteiro para a classe base, apontando para um objeto derivado. Aqui existe o custo de mais um ponteiro. Exemplo de chamada: **ptr->metodo()**, onde ptr é o ponteiro do tipo base.

Além destas condições, foram empregados três formatos diferentes:

1. **Função (método) sem argumentos e sem retorno;**

2. **Função (método) com 1 argumento e sem retorno** e

3. **Função (método) com 1 argumento e 1 retorno.**

Uma forma de otimização alcançada ainda no código fonte é produzida ao se especificar uma função como **in-line**, causando a cópia do corpo da função em cada local de chamada, eliminando o tempo adicional gasto com a troca de contexto. Este efeito no entanto só foi possível de ser observado na compilação otimizada.

Os seguintes gráficos apresentam os resultados obtidos através da compilação e execução do código incluído no anexo. O gráfico 6.1 apresenta o resultado do código compilado sem nenhuma otimização. O gráfico 6.2 apresenta a execução do código compilado com otimização em velocidade.

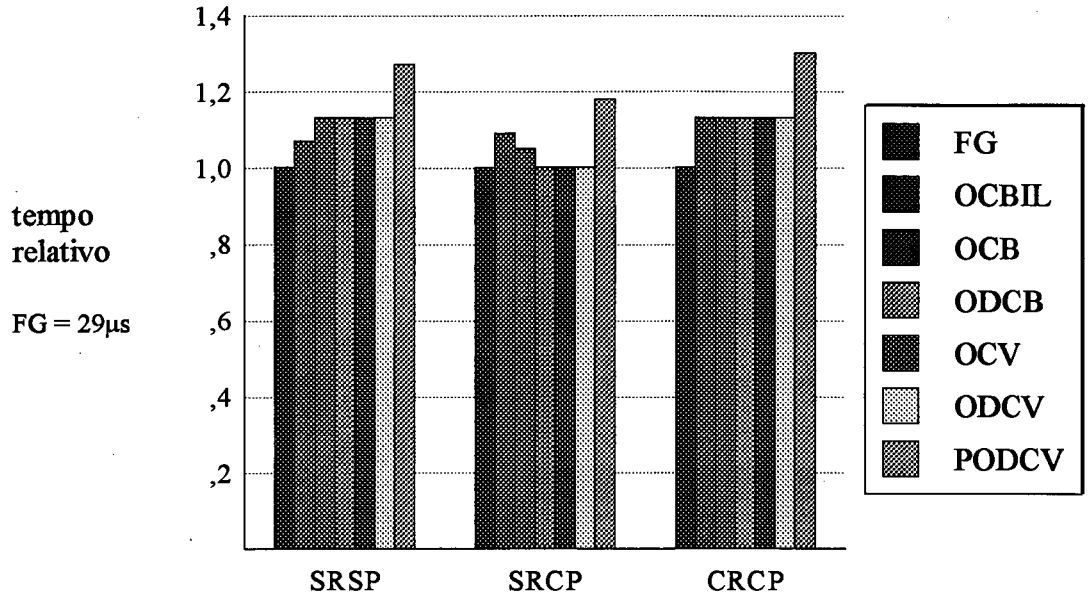


Gráfico 6.1 - Resultados produzidos com código compilado sem otimizações.

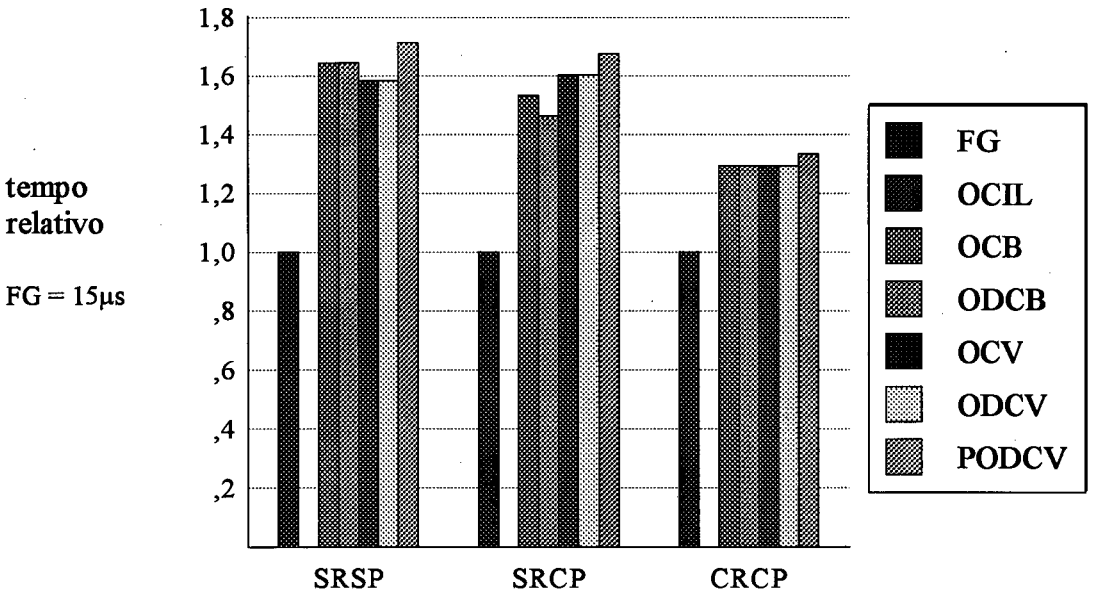


Gráfico 6.2 - Resultados produzidos com código compilado com otimização em velocidade.

Legendas :

- FG - chamada de função global.
- OCBIL - chamada de função inline em classe sem funções virtuais.
- OCB - chamada de função em classe sem funções virtuais.

- ODCB - chamada de função da classe base através de objeto de classe derivada sem funções virtuais.
- OCV - chamada de função de classe base declarada como virtual.
- ODCV - chamada de função virtual através da classe base através de objeto derivado.
- PODCV - chamada de função virtual da classe base, através de ponteiro do tipo da classe base apontando para objeto derivado.
- SRSP - funções sem retorno e sem parâmetro de chamada
- SRCP - funções sem retorno e com um parâmetro de chamada
- CRCP - funções com retorno e com um parâmetro de chamada

A instanciação de cada classe foi efetuada junto à declaração da função global, respeitando o escopo de chamada. Se por outro lado estas instanciações são feitas dentro da função **main()**, os tempos de chamadas dos métodos se reduzem substancialmente.

7 - Objeto-Imagem

7.1 - Introdução

Este capítulo utiliza a técnica de identificação e modularização dos elementos que interagem com o meio externo de um sistema (sensores e atuadores) proposta por LÜCKE & DE NEGRI (1993), detalhando formas de transposição destes para o meio computacional utilizando o paradigma de orientação por objetos através do artifício do **objeto-imagem**, descrito preliminarmente em LÜCKE, DE NEGRI & MADEIRA (1995). Esta técnica tem por objetivo facilitar a construção de modelos e posterior transposição para a codificação, utilizando a orientação por objetos na abstração dos elementos e construção de suas interações.

7.2 - Identificação dos sistemas de atuação (SA) e sistemas de medição (SM)

Segundo SMITH & GAWTHROP (1992) o projetista deve ser capaz de trabalhar em um alto nível de abstração, utilizando uma notação que forneça um mapeamento próximo do sistema físico a ser representado, sendo preferível o uso de uma notação gráfica que preserve a estrutura do sistema. Neste sentido, o trabalho de LÜCKE e DE NEGRI (1994) favorece a descrição de um sistema automático, onde a modelagem é caracterizada pela identificação dos sistemas de atuação e medição e dos recursos básicos presentes de matéria, energia e informação, segundo exposto no capítulo 1. Esta divisão permite uma modularização que facilita a identificação dos elementos que devem interagir com o meio computacional e fornecer abstrações para este.

Para a apresentação do **objeto-imagem** utiliza-se como exemplo uma instanciação para o canal de atuação SA do modelo central de um sistema automático[17], apresentado pela figura 7.1 na forma de um diagrama esquemático. Este detalhamento servirá como base para a criação das abstrações no sistema de informação (meio computacional).

---

[17] Figura 2.3 no capítulo 2 - Sistemas Automáticos.

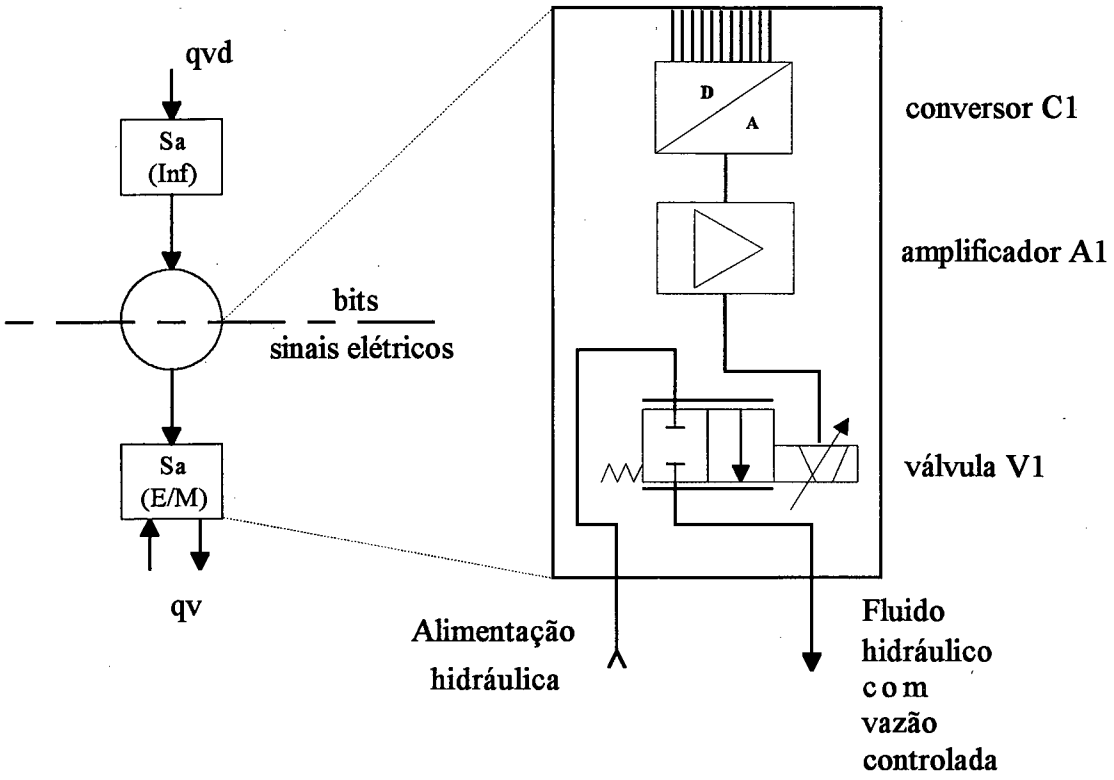


Figura 7.1 - Instanciação da parte física de uma SA em vazão

Este exemplo configura um **sistema de atuação em vazão** que opera segundo informações provenientes de um computador. O conversor D/A C1 traduz o sinal digital produzido no computador para uma tensão compreendido entre 0 a 10 volts. O amplificador de potência A1 adapta esta tensão para a corrente de pilotagem de uma válvula que se situa entre 0 e 3A. Uma válvula proporcional V1 é escolhida para o controle da grandeza vazão (qv) no meio físico/químico na faixa de 0 a 50 l/s.

A alimentação hidráulica é o recurso de energia/matéria que sofre transformação pelo sistema. O recurso de informação é fornecido pelo computador. A figura 7.2 esquematiza este circuito através do detalhamento na notação canal agência, incluindo as grandezas e suas faixas de variação.

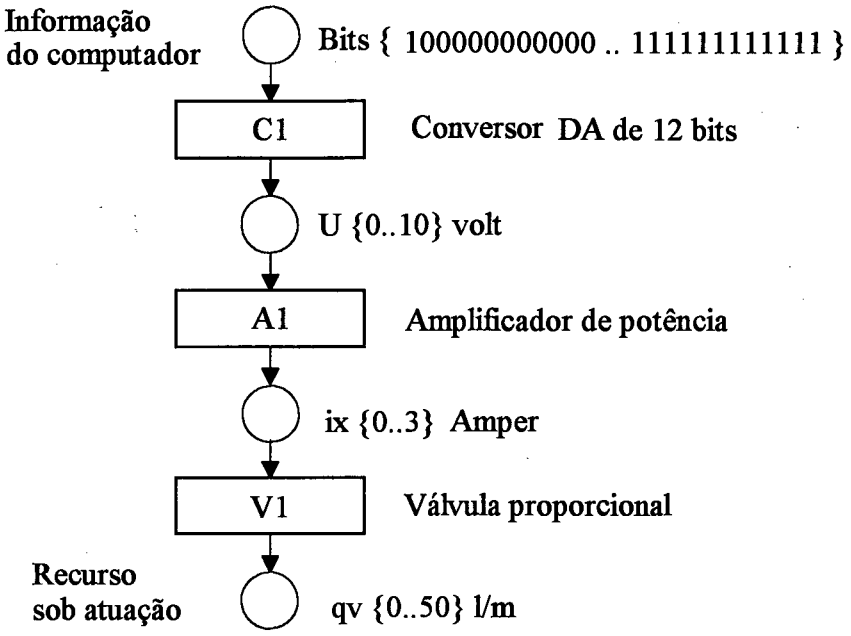


Figura 7.2 - Modelo canal/agência da parte física do sistema de atuação em vazão.

7.3 - Abstração

Uma das formas de representar o modelo no meio computacional é utilização da orientação por objetos. Através de um espelho conceitual, cada elemento do mundo real é transposto para um objeto na solução computacional, mantendo relações simétricas entre estes e os elementos físicos do sistema. (figura 7.3)

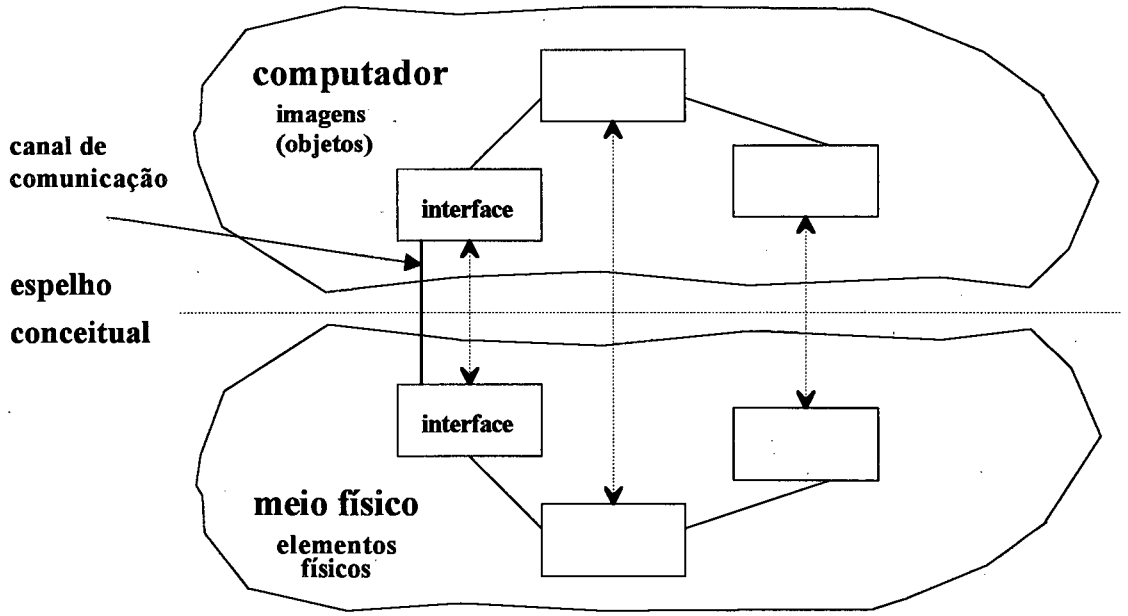


Figura 7.3- Espelho

A adição de canais de comunicação entre computador e o meio físico (interfaceamento), proporciona a observação e alteração do estado dos elementos reais através de um conjunto de abstrações no computador, permitindo que o sistema de informação se comunique com os elementos do meio energético/material.

7.4 - Estado

É possível diferenciar um elemento físico de outro através do conjunto de seus atributos. Como exemplo de atributos tem-se temperatura, velocidade, viscosidade, a vazão de uma válvula, a rotação e torque de um motor, etc. Na criação de uma abstração para o computador, é escolhido um subconjunto dos atributos do elemento real.

A observação da variação do conteúdo destes atributos fornece uma descrição das condições atuais do elemento. Esta descrição é denominada **estado atual**, ou simplesmente **estado** do elemento. No computador esta abstração é representada por um **objeto**, capaz de manter um **estado**. Segundo BOOCH (1991) :

> *"O estado de um objeto engloba todas as suas propriedades (usualmente estáticas) mais o valor atual (usualmente dinâmico) de cada uma de suas propriedades."*

Para que o objeto possa armazenar seu estado, é escolhida uma estrutura de dados representativa. Em um sistema técnico, tem-se por exemplo a variável **qv**, que armazena a vazão ajustada para uma válvula (fig. 7.4) .

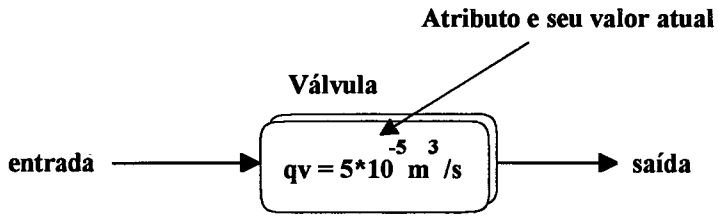


Figura 7.4 - Atributo e seu valor atual para representação da vazão ajustada de uma válvula

7.5 - Função de transferência

Para que haja modificações no estado de um objeto são enviadas mensagens que efetuam operações sobre este. Entre estas operações é possível distinguir aquela que ativa o comportamento principal que se deseja abstrair, a qual neste trabalho é denominado **função de transferência** (FT) (Fig. 7.5).

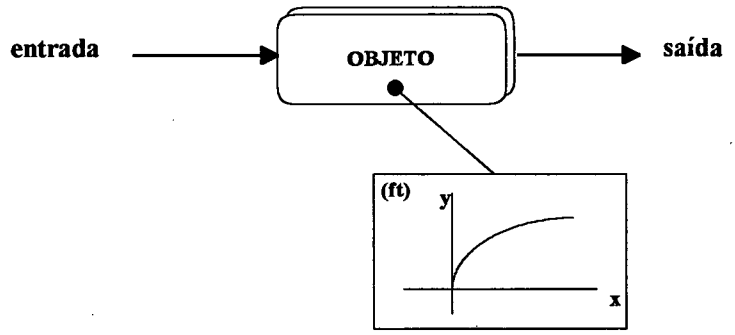


Figura 7.5 - Objeto e função de transferência associada

As operações, ou mensagens, sobre um objeto fazem com que este altere o conteúdo de um ou mais atributos, o equivalente ao armazenamento de novos valores para sua estrutura de dados, levando o objeto a um novo estado (figura 7.6) (WEGNER, 1989).

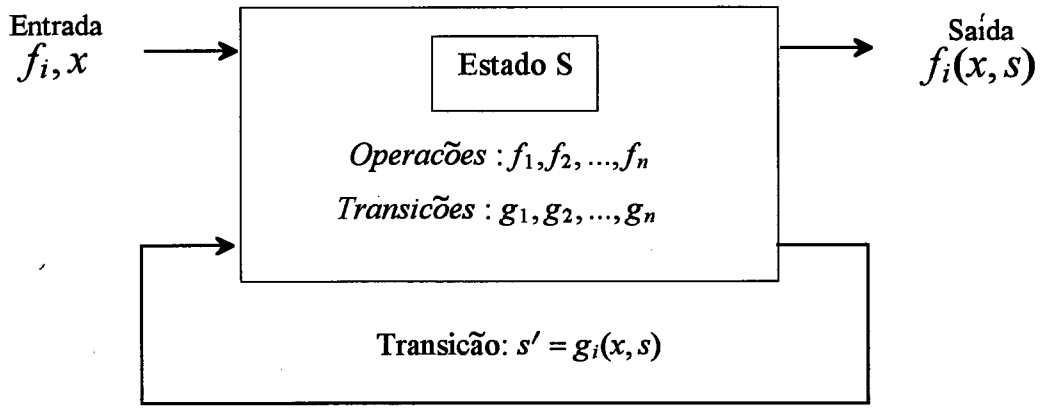


Figura 7.6 - Objeto, operações e estados (WEGNER, 1989).

Para a existência do objeto no meio computacional são ainda necessárias operações e dados auxiliares próprios ao domínio da implementação.

7.6 - Comunicação entre objetos

Um elemento no meio físico interage com outro elemento através do controle de certas grandezas, inerentes ao meio que estabelece a comunicação. No exemplo da parte física do sistema de atuação em vazão (fig. 7.1), o envio da informação do amplificador para a válvula é feito através da variação da corrente elétrica presente nos condutores que os interligam. No meio computacional esta informação é transferida de um objeto a outro através de mensagens. Assim como um objeto abstrai sua contraparte no meio físico, um **link** abstrai uma ligação entre dois objetos. Aqui também é feita uma seleção de quais características da ligação são relevantes para a solução computacional.

7.7 - Interfaceamento

Além da criação de objetos e a comunicação entre estes, se faz necessário a comunicação entre os objeto e sua contraparte real, o que é fornecido através de dispositivos e rotinas de interfaceamento. A figura 7.7 ilustra este arranjo genérico através da notação canal/agência.

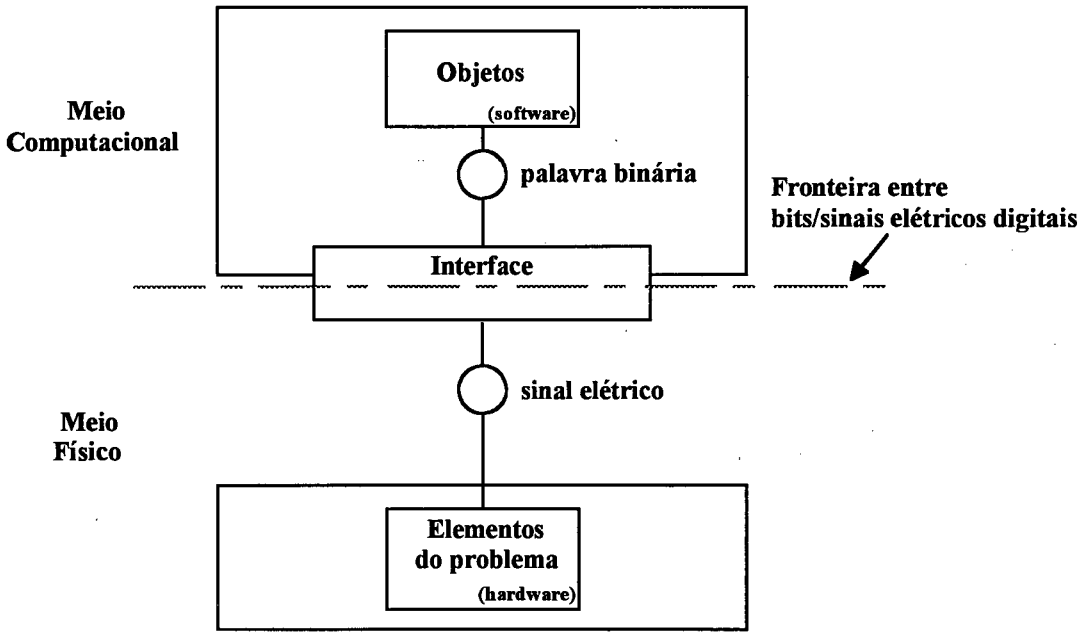


Figura 7.7- Comunicação entre objeto e sua contraparte física.

7.8 - Abstração

Para que os elementos do meio físico possam ser abstraídos para o meio computacional e além disto mantenham canais de comunicação com estes, utiliza-se o **objeto-imagem**. A figura 7.8 utiliza o exemplo do sistema de atuação em vazão, integrando o diagrama do modelo canal/agência da parte física (fig. 7.2) com a representação destes pelo artifício do **objeto-imagem**.

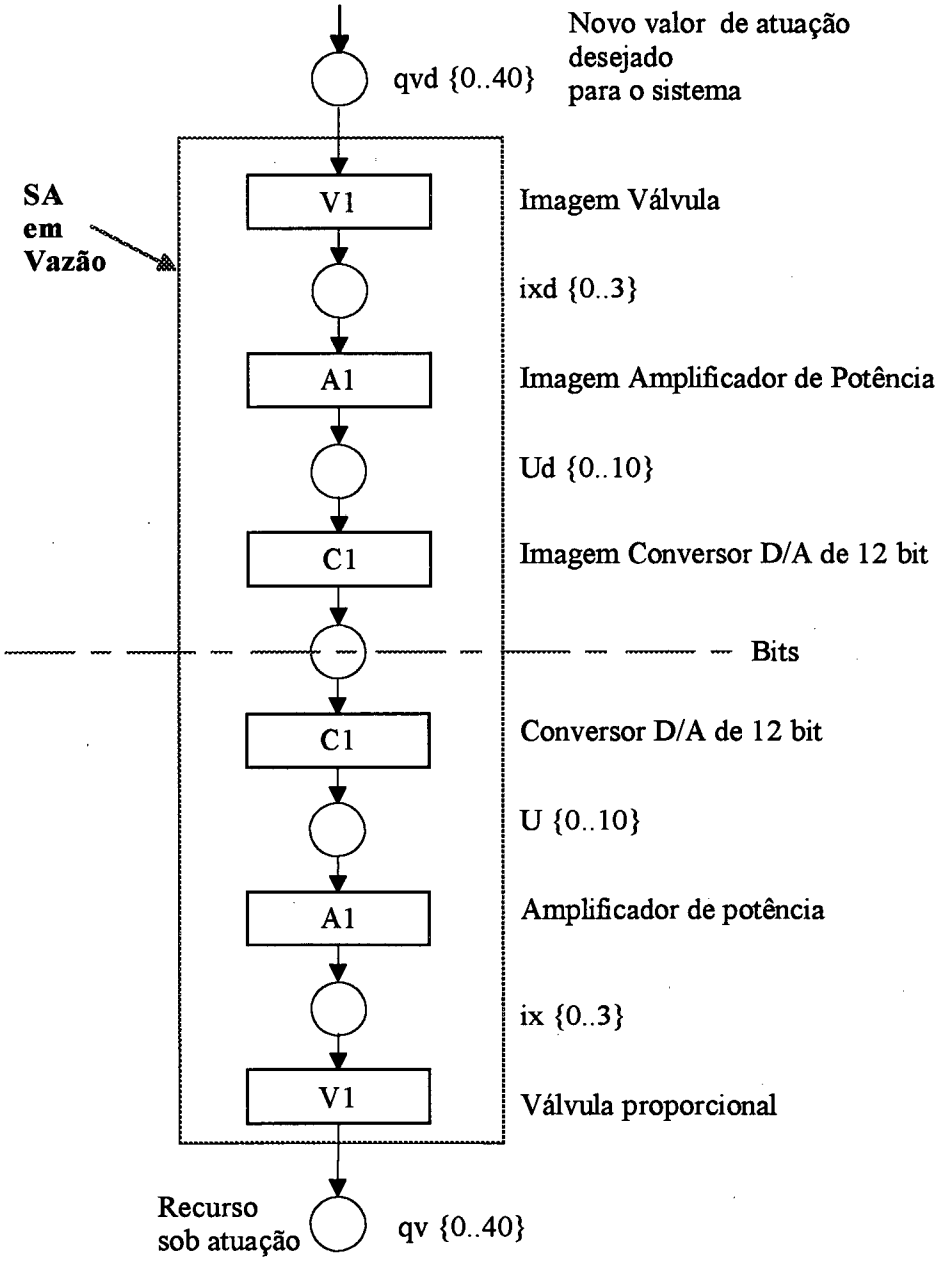


Figura 7.8 - Sistema de atuação em vazão, incluindo a parte presente no meio computacional, compondo a visão comportamental do sistema.

Cada canal contém a informação sendo transmitida de elemento a elemento. O valor desejado de atuação do sistema é recebido do exterior (possivelmente um sistema de coordenação de tarefas), fazendo com que o estado do SA em vazão como um todo seja alterado.

7.9 - Caracterização do objeto-imagem

O **objeto-imagem** se diferencia do objeto tradicional no sentido em que não representa um modelo de simulação. Cada objeto imagem possui uma função de transferência simétrica a sua contraparte real, o que lhe fornece um comportamento simétrico. No exemplo do sistema de atuação em vazão, a válvula real recebe uma corrente de entrada e produz uma vazão correspondente em função de sua curva característica. No **objeto-imagem** esta função é invertida. De forma prática, o objeto imagem **V1** do exemplo irá calcular qual a corrente necessária a ser aplicada à sua contraparte real no meio físico, para que esta forneça a vazão desejada para o sistema hidráulico. No exemplo o mesmo raciocínio se aplica aos outros componentes do modelo. Como conseqüência, a relação entrada/saída também é invertida, permitindo que a mensagem enviada para o objeto seja aquela que contém o novo estado desejado para este, e não um valor de atuação sobre o elemento. Como efeito, o utilizador do **objeto-imagem** trabalha em um nível maior de abstração[18], isolando-se dos detalhes de comando dos elementos.

A figura 7.9 apresenta um modelo para um sistema de atuação em vazão, associando as curvas de operação para cada elemento. Estas curvas são fornecidas aos objetos-imagem no momento de sua instanciação, ligando cada objeto à sua função de transferência específica[19].

---

[18] Este efeito se contrapõe à prática normal onde o utilizador fornece o valor de entrada que simula o elemento real. No caso da válvula do exemplo, seria fornecida a corrente de pilotagem, suprimindo-se o nível de abstração que se pretende com o modelo empregando o **objeto-imagem**.

[19] Este esquema possibilita a criação de bibliotecas de funções de transferência que podem ser reutilizadas para objetos diferentes.

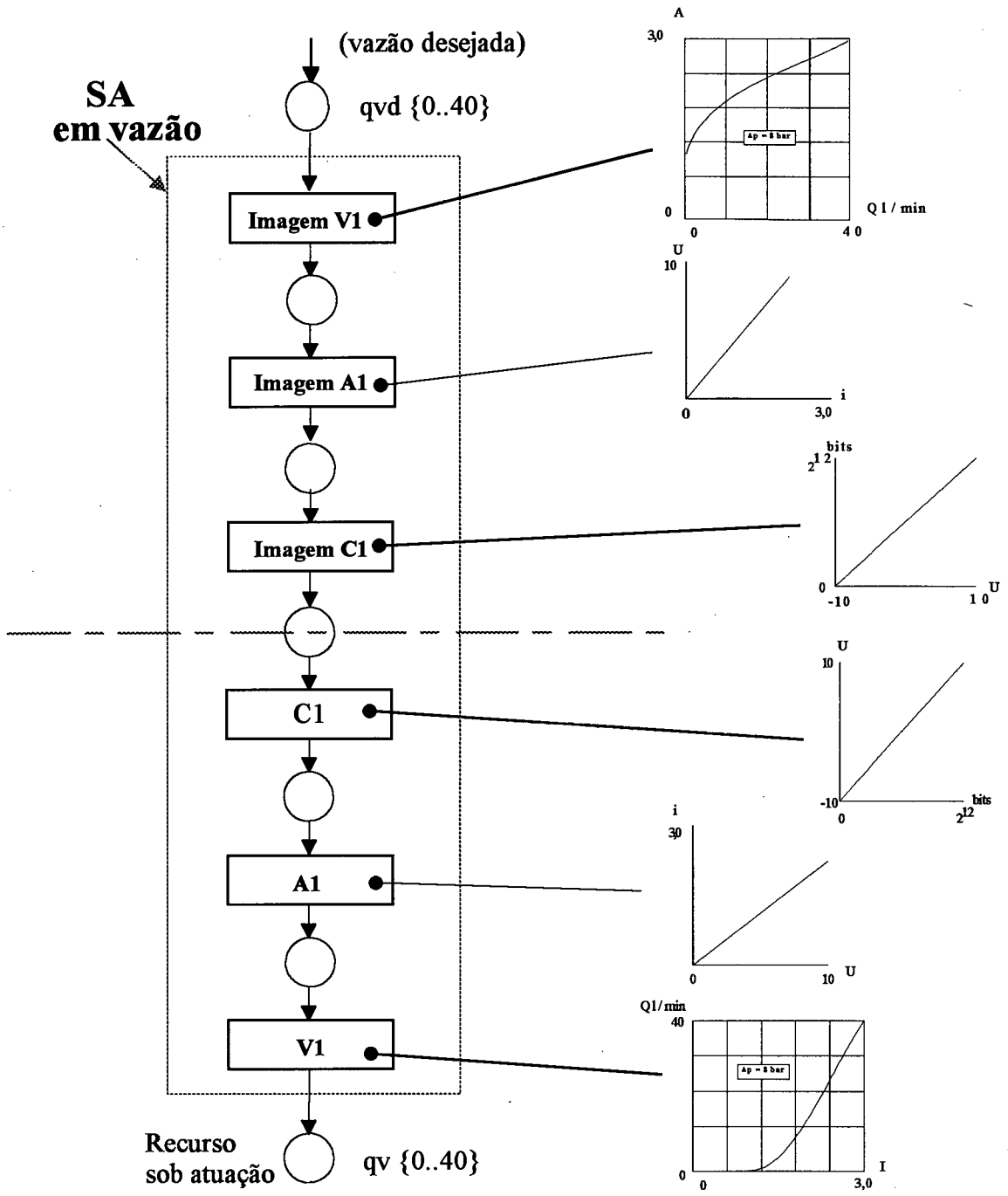


Figura 7.9- Sistema de atuação em vazão e curvas dos objetos.

Existe a possibilidade de agrupamento de elementos no modelo quando as funções de transferência individuais não são fornecidas pelo fabricante, mas sim para um bloco, como por exemplo um amplificador e válvula de um mesmo fabricante[20], onde geralmente é fornecida a vazão em função da tensão aplicada ao conjunto.

---

20 Exemplificado no capítulo 3, figura 3.21

A construção de um modelo para um sistema discreto envolve o fornecimento de uma tabela verdade para cada objeto, produzindo relações semelhantes[21].

7.10 - Implementação do objeto-imagem

Cada **objeto-imagem** é descrito por uma classe. A figura 7.10 representa um modelo dos relacionamentos encontrados no SA vazão.

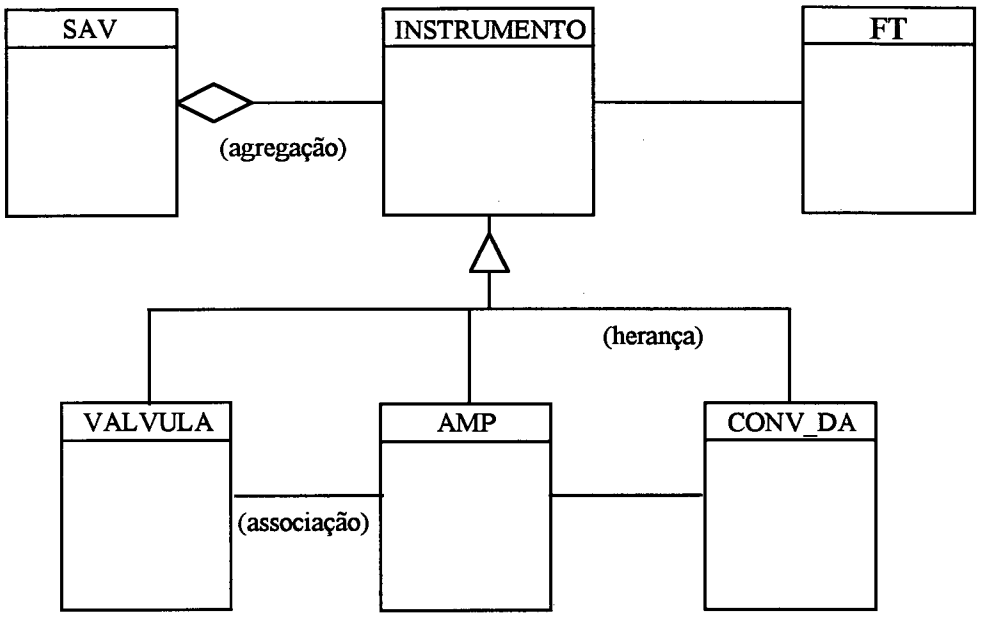


Figura 7.10 - Descrição dos relacionamentos entre as classes do **SAV**

Nesta estrutura a classe **SAV** é composta de elementos do tipo INSTRUMENTO, que por sua vez podem ser uma válvula (classe **VALVULA**), um amplificador (classe AMP) e um conversor D/A (classe CONV_DA). As associações entre VALVULA e AMP, como também entre AMP e CONV_DA, indicam os links que irão existir no momento de instanciação das classes. A classe FT fornece a função de transferência para o instrumento.

A classe FT serve de base para uma hierarquia que contém diversos tipos de funções de transferência, resultando na possibilidade de reutilização destas para objetos que se comportam de maneira semelhante. Esta estratégia também permite que constantes e variáveis auxiliares não sejam representadas nas classes dos instrumentos, tornando-as mais genéricas.

---

[21] Exemplificado no capítulo 8

8 - Exemplificação de Utilização do Objeto-Imagem

8.1 - Introdução

Este capítulo emprega como exemplo de aplicação do **objeto-imagem** o modelo de uma **unidade de potência e condicionamento hidráulico**[22] **(UPCH)** organizado em subsistemas de atuação e medição, utilizando a notação canal agência como meio de representação. Dois subsistemas são escolhidos e decompostos em imagens e elementos reais. Como produto obtém-se a descrição detalhada de dois subsistemas na notação canal/agência, um diagrama de fluxo de objetos - OFD (ELLIS, 1994) e exemplos de codificação em C++. Complementando a visão do problema, também é ilustrada hierarquias parciais de classes resultantes da construção das abstrações para o trabalho em desenvolvimento.

8.2 - Unidade de potência e condicionamento hidráulico - UPCH

A finalidade da unidade de potência e condicionamento hidráulico é fornecer alimentação de fluido hidráulico com temperatura, pureza, vazão, pressão e direcionamento controlados para a alimentação de circuitos hidraúlicos. O sistema visa atender diversas configurações e regimes de funcionamento, de acordo com a natureza dos sistemas alimentados por suas saídas. O gerenciamento de suas atividades é exercido por um sistema computadorizado. A figura 8.1 ilustra o modelo central de fluxo de energia/matéria da unidade através da notação canal/agência.

---

[22] Atualmente em fase de testes iniciais de funcionamento no Laboratório de Hidráulica e Pneumática da UFSC (LASHIP).

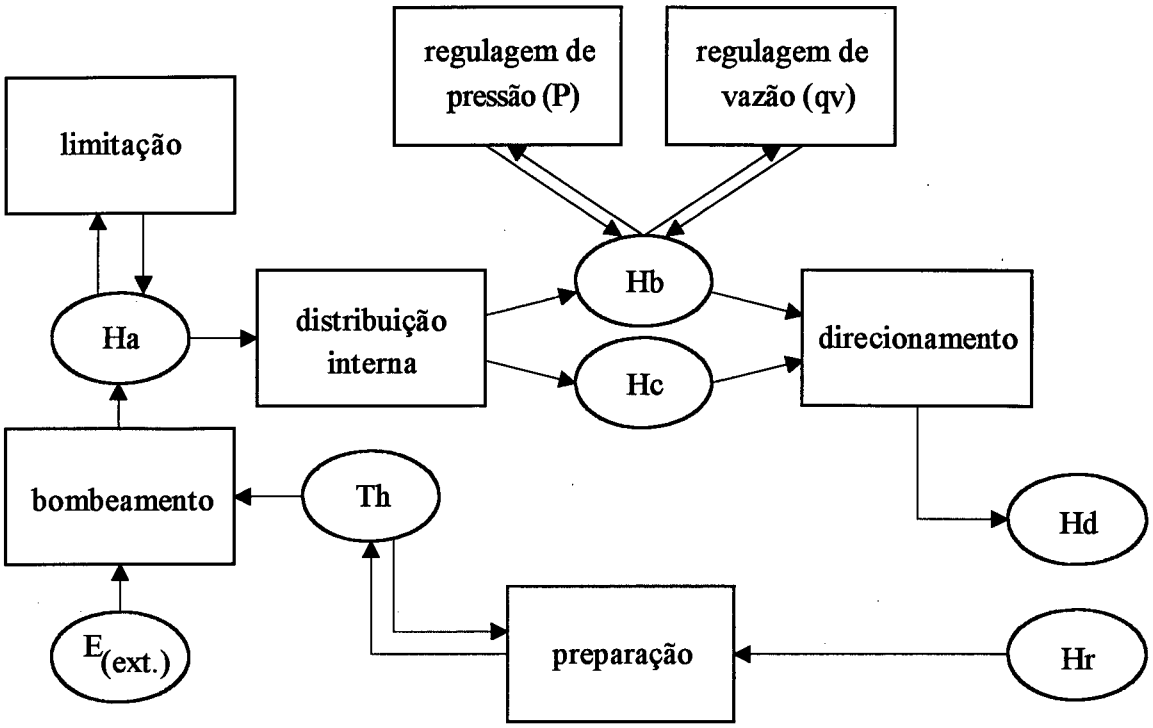


Figura 8.1 - Modelo central da Unidade de Potência e Condicionamento Hidráulico (UPCH).

Os retângulos representam as unidades de transformação presentes na UPCH, classificadas como sistemas de atuação (SA), sistemas de medição (SM) ou blocos com as duas operacionalidades (SA/SM). Os círculos representam recursos de energia/matéria, como eletricidade e fluido hidráulico. As unidades de transformação possuem canais de comunicação (não ilustrados na figura 8.1) para a troca de informações com um computador, que tem o papel de gerenciar a unidade de potência e condicionamento hidráulico.

Descrição das unidades de transformação:

a) Preparação - controla a temperatura e pureza do fluido hidráulico. Estes parâmetros atuam como pré-condição de funcionamento da unidade;

b) Bombeamento - a partir dos recursos de fluido hidráulico, proveniente do tanque de armazenamento (Th), e da energia elétrica externa ($E_{ext.}$), fornece vazão e pressão para os outros elementos da unidade;

c) Limitação - controla a pressão máxima de funcionamento interno do circuito hidráulico e fornece sua monitorização;

d) Distribuição primária - controla o direcionamento do fluxo do fluido hidráulico nos circuito internos da unidade;

e) Regulagem em pressão - controla a pressão de saída da unidade (P);

f) Regulagem em vazão - controla a vazão de saída da unidade (qv);

g) Direcionamento - sistemas de direcionamento do fluido hidráulico para as tomadas externas.

Descrição dos recursos:

a) **Hr** - tomadas de retorno do fluido, após a passagem pelo(s) circuito(s) externos sendo alimentados pela unidade;

b) **Ht** - depósito do fluido hidráulico;

c) **E(ext.)** - energia elétrica para alimentação do bombeamento;

d) **Ha** - fluido hidráulico com pressão e vazão adequados para alimentação interna dos circuitos da unidade;

e) **Hb** - fluido hidráulico com pressão e vazão controlados;

f) **Hc** - fluido hidráulico com pressão e vazão não controlados;

g) **H1** - tomadas de saída da unidade.

A figura 8.2 esquematiza como é feita a interação entre as unidades de transformação e o gerenciamento exercido pelo computador.

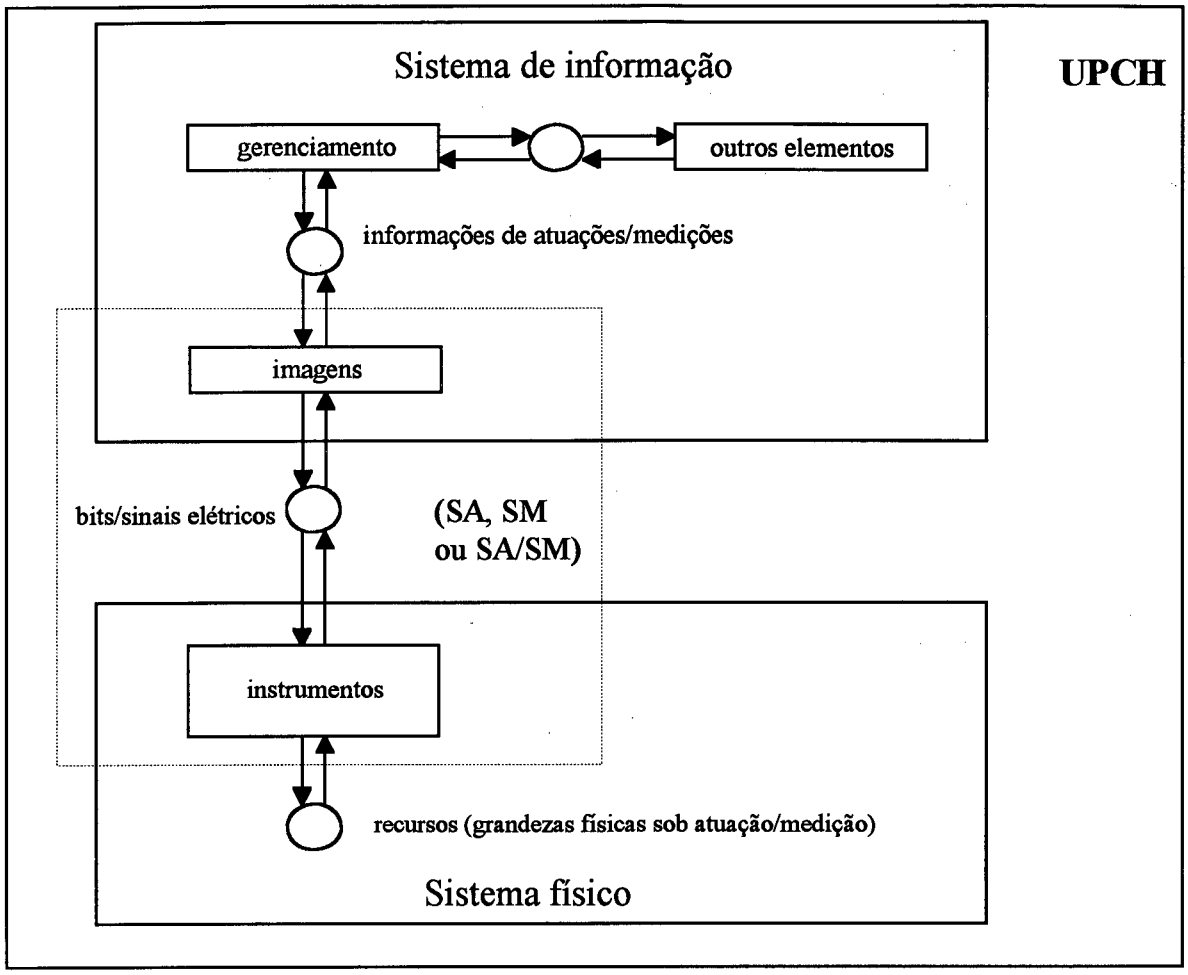


Figura 8.2 - Comunicação entre o computador e a UPCH.

Cada sistema SA, SM ou SA/SM é composto de instrumentos (válvulas, amplificadores, relés, interfaces, etc.) e suas imagens no computador. O gerenciamento envia mensagens para estes sistemas, informando o novo valor desejado de atuação, leitura de medições, e também troca de mensagens com os outros elementos do sistema de informação (interface gráfica e outros elementos auxiliares).

8.3 - Sistema de atuação em vazão - SAV

A figura 8.3 detalha o sistema **f** da figura 8.1. Neste caso o sistema é apenas de atuação (SA), não comportando instrumentos de medição. O gerenciamento envia uma mensagem de entrada para o SA informando o novo valor de atuação desejado de vazão **qvd**. O sistema então

estabelece um novo estado atual introduzindo a modificação na grandeza qv sob atuação no sistema energético/material.

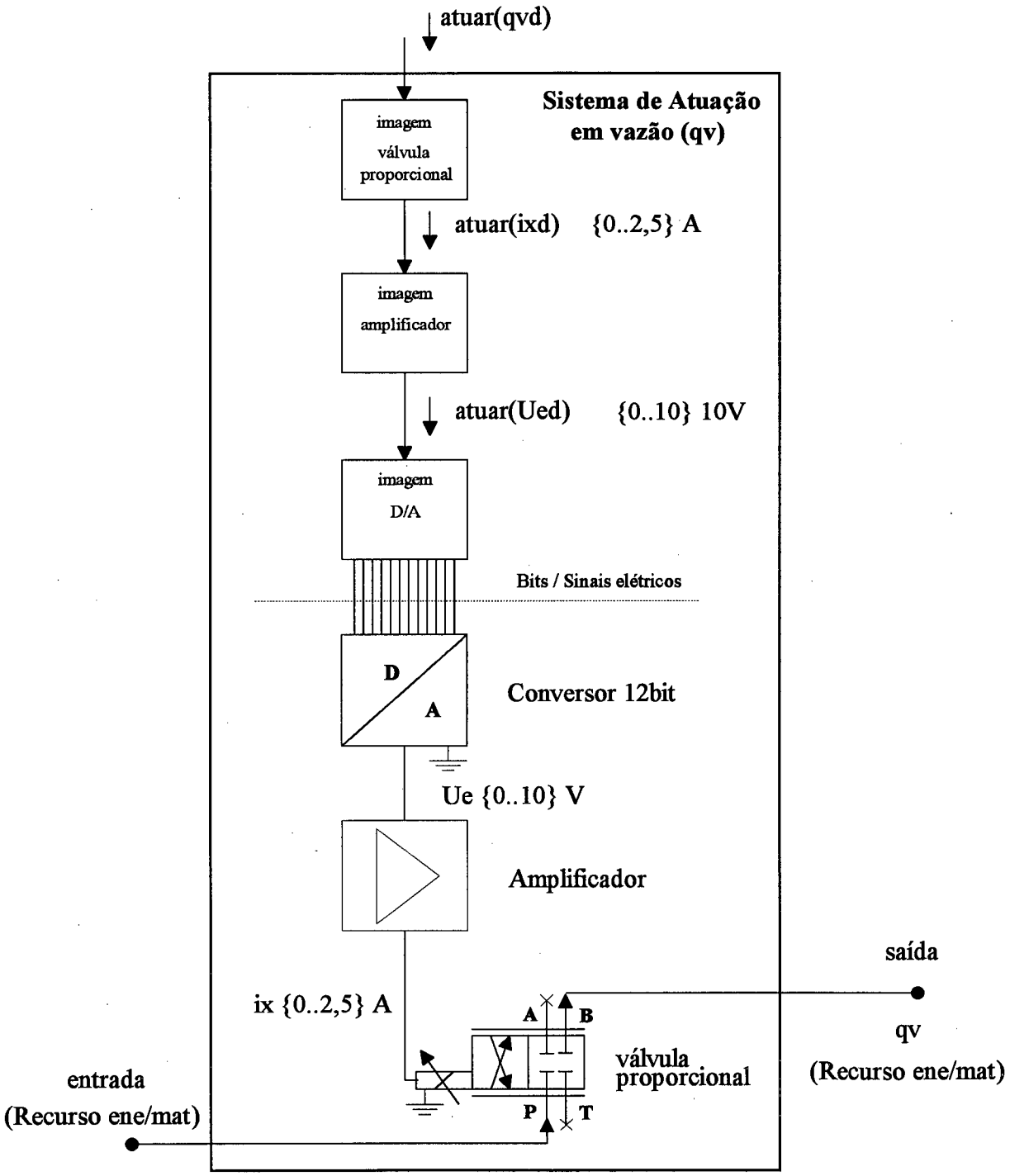


Figura 8.3 - Sistema de Atuação em vazão

O comportamento do sistema é tal que um novo valor desejado para a grandeza sob atuação seja efetivamente o valor fornecido pelo gerenciamento, encapsulando do utilizador os detalhes das transformações internas do sistema. Cada **objeto-imagem** é responsável em

computar o valor que deve ser fornecido a sua contraparte no meio físico por ele abstraída. Desta forma o SAV possui em sua parte no sistema de informação um conjunto de objetos que estão interligados de uma forma simétrica aos elementos presentes no meio físico.

Para a codificação da classe SAV em C++ tem-se o seguinte exemplo:

```
class SAV
{
        char *id;  // guarda sua identificação
        FT *pft1, *pft2;        // ponteiros para funções de
                                // transferências dos instrumentos
                                // utilizados
        INSTRUMENTO *pamp, *pvalvula;  // ponteiro para os instrumentos
        double valor_atual;  // valor atual do SAV
        double saida_valvula, saida_amplificador; // Variáveis
                                                  // auxiliares

public:
        SAV(char *ID );
        ~SAV(){ delete pamp, pvalvula };
        void atuar(double vazao_desejada); // atua com a nova vazão
                                           // desejada
        double estado() {return vazao_atual};
        atualizar();
};
```

Definição dos métodos da classe SAV:

```
extern CONV_DA conv_da_SAV;   // o conversor é declarado global em
                              // outro módulo
// contrutor
SAV::SAV(char *ID)
{
        // Associa uma identificação para este sistema de atuação
        id = ID;
        // Cria uma função de transferência para a válvula
        pft1 = new FT_1("ft1",31.778,1.23816e-4);
        // Cria uma função de transferência para o amplificador      // para a servoválvula
(ax + b)
        pft2 = new FT_1("ft2",500,0);
        // Cria válvula, com corrente e tensão máximas de entrada,
        // função de transferência e referência para ligação ao
        // conversor
pamp = new AMPLIFICADOR("amp1", 20, 24, pft2, &conv_da);
        // Criação da válvula, forncecimento da vazão máxima, corrente
        // máxima, função de transferência e referência para o
        // amplificador
        pvalvula = new V_PROPORCIONAL("Val. Prop. v1", 50e-5, 3,5,
                                      pft1, pamp);

        // zera a vazão da válvula
        atuar(0);
}
```

```
SAV::~SAV()
{
        // libera memória
        delete pft1,pft2,pamp,pvalvula;
}

void SAV::atuar(double vazao_desejada)
{
        saida_valvula = pvalvula->entrada(vazao_desejada);
        vazao_atual=vazao_desejada;
}
```

8.4 - Sistema de atuação e medição em direcionamento de vazão manual - SASMDR

A figura 8.4 ilustra uma primeira aproximação para a construção de um sistema em **direcionamento**, desta vez com características de atuação e medição conjuntas (SASM), estabelecendo as informações presentes no sistema, direcionamento das mensagens e sentido da informação. A válvula empregada para o controle no meio físico é de atuação manual, sendo portanto necessário a inclusão de uma interface homem/máquina para o fechamento do elo de comunicação. A detecção do posicionamento final da válvula é obtido através de duas chaves fim-de-curso instaladas nos batentes da manopla de atuação da válvula.

Com esta visão em blocos e esquema de ligações é possível identificar os elementos que farão parte da solução computacional. Cada mensagem no meio computacional equivale a transferência de um sinal no meio físico. Como é possível verificar, mesmo a transmissão de sinais não elétricos podem ser representados, no caso a informação para o operador e sua intervenção no sistema.

Na solução computacional aparecem elementos que não fazem parte da solução física, neste exemplo os métodos **medição()** e **comparação()**. A mensagem de atuação desejada para o sistema é recebida através do método **comparação** que aciona medição e gera o erro resultante. Caso a atuação desejada não seja igual ao estado corrente do sistema, é enviado para o **objeto-imagem** da válvula o estado desejado para esta, computando a posição necessária em sua entrada (HORARIO, ANTI-HORARIO}, desencadeando a cascata de mensagens necessárias até o acionamento da válvula no meio físico.

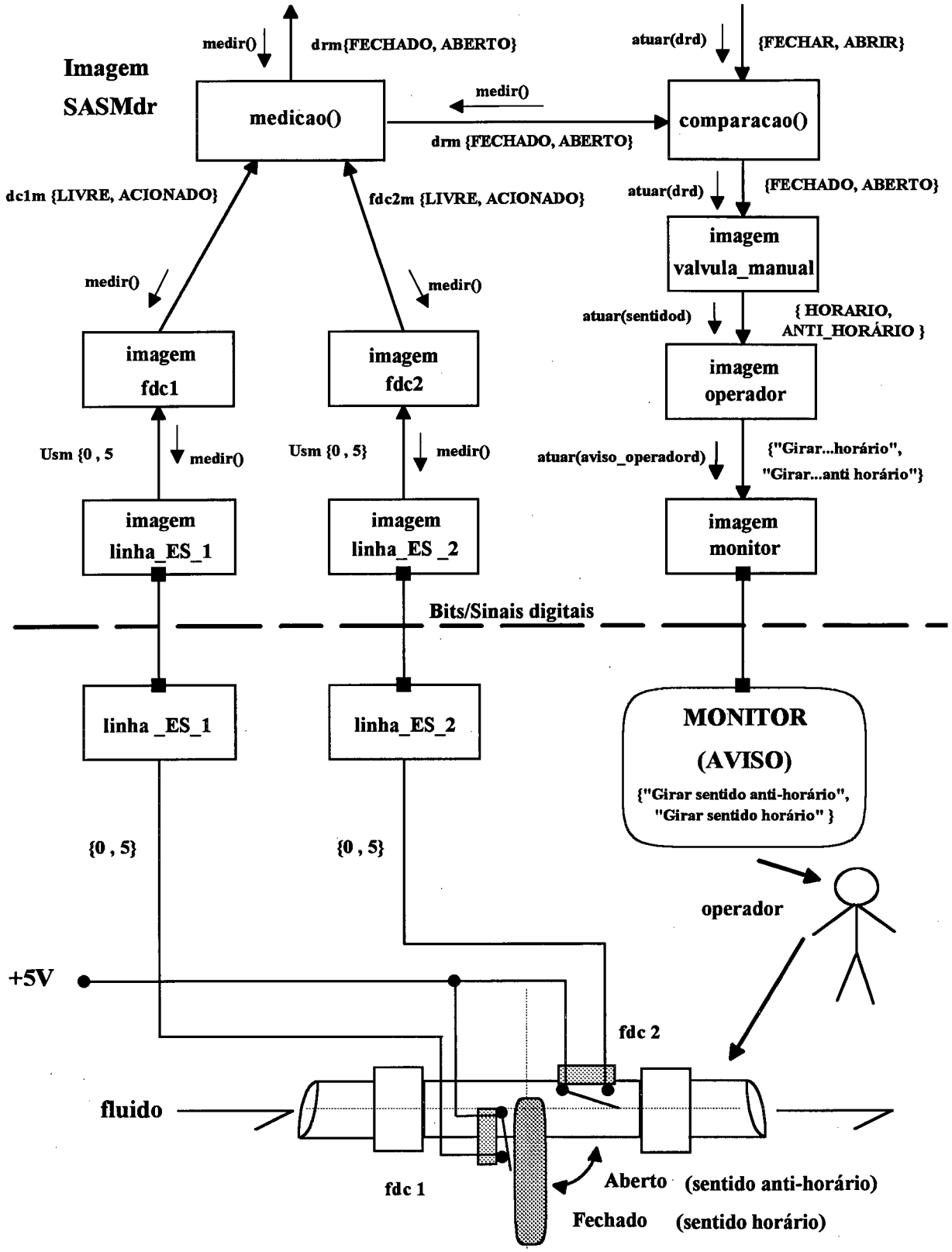


Figura 8.4 - Sistema de atuação/medição em direcionamento

Como exposto anteriormente, todas as classes possuem uma ligação com a função de transferência que descreve seu funcionamento. No entanto, quando esta função é muito simples, como no caso do comportamento de **fdc1** e **fdc2** do exemplo, mapeando {0, 5 } em {LIVRE, ACIONADO}, ela é dispensada. A figura 8.5 apresenta o OFD (ELLIS, 1994) para o projeto detalhado da solução computacional. A partir de simplificações lógicas elimina-se algumas classes/objetos, evitando a chamada de alguns métodos, o que otimiza o desempenho no tempo de execução do código a ser gerado.

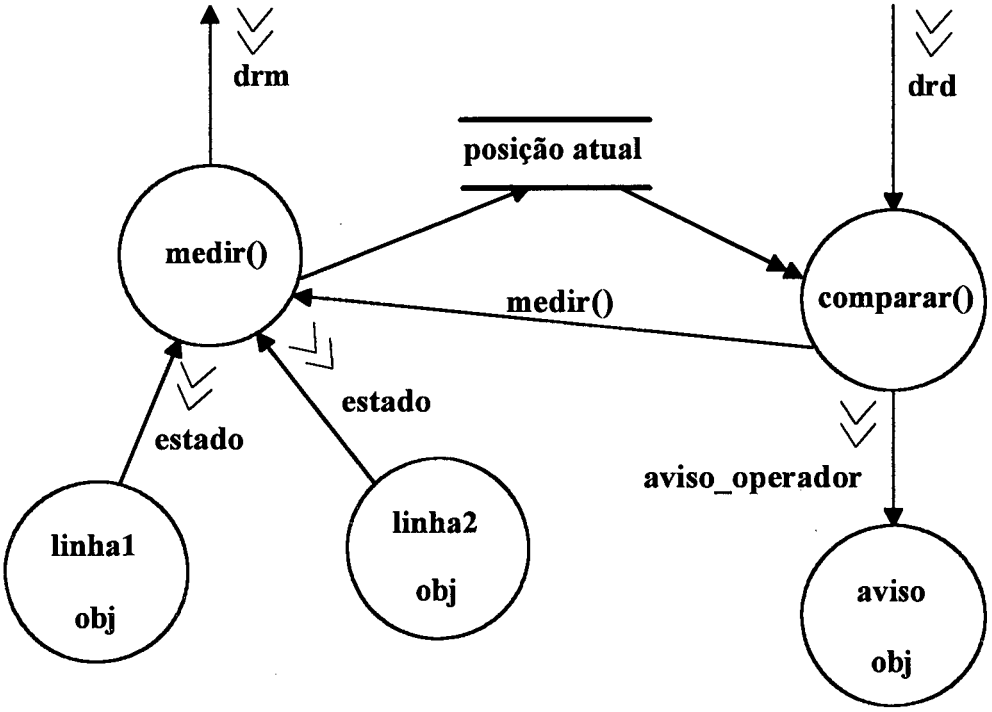


Figura 8.5 - OFD para o exemplo do SASM em direcionamento

Existem dois canais de comunicação com a classe/objeto SASMDR: o primeiro recebe a direção desejada para a vazão (drd); o segundo fornece a direção medida (drm). As duas setas (>>) indicam a origem da mensagem que iniciou o fluxo de informação. Os círculos que contém a indicação **obj** são instâncias de classes utilizadas para a composição do sistema. Os demais círculos representam métodos da classe SASMDR. O depósito de dados **posição atual** armazena o estado atual do sistema.

Além dos elementos presentes na figura 8.5, o sistema pode contar ainda com mensagens para atualização da atuação, para manuseio de falhas (como o posicionamento incorreto da manopla) e operações para verificar a compatibilidade de ligação entre objetos do tipo SISTEMA no momento da instanciação. A figura 8.6 ilustra a hierarquia para um pequeno conjunto de classes para sistemas discretos.

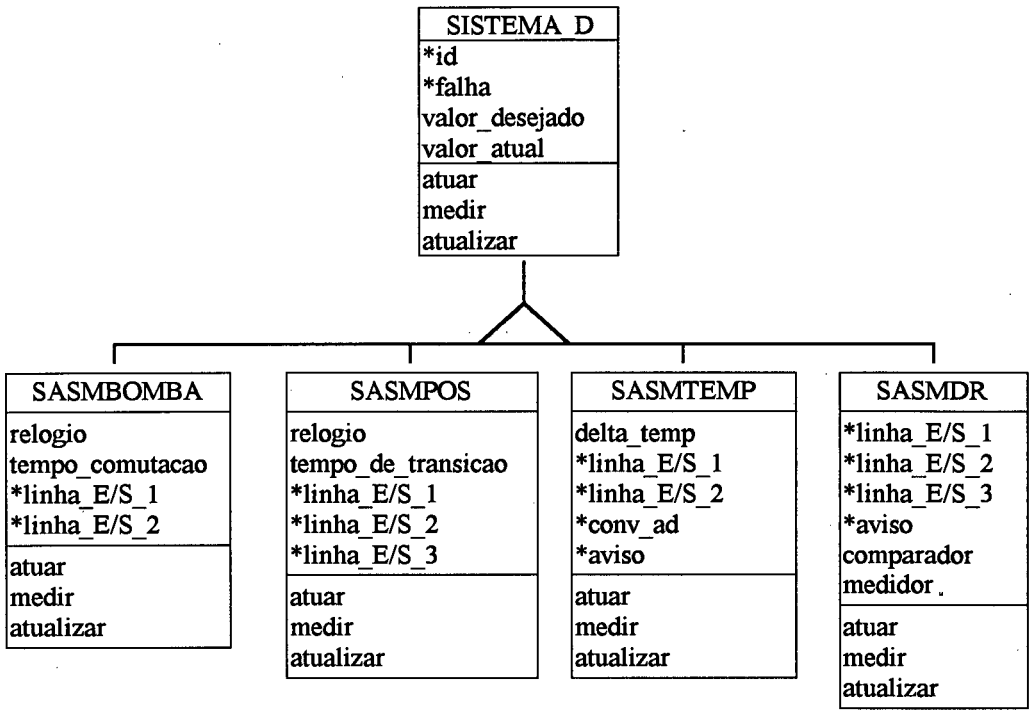


Figura 8.6 - Hierarquia para o conjunto de sistemas discretos do exemplo.

A hierarquia é mantida polimórfica para simplificação do gerenciamento e facilidade de extensões. A figura 8.7 ilustra a hierarquia para elementos que compõem os sistemas. A classe INSTRUMENTO possui uma referência para si mesma (figura 8.8), o que permite a qualquer objeto derivado se conectar a outro da mesma hierarquia. Esta ligação é posteriormente verificada no momento de criação do objeto (método construtor), evitando a representação de conexões não compatíveis no meio físico.

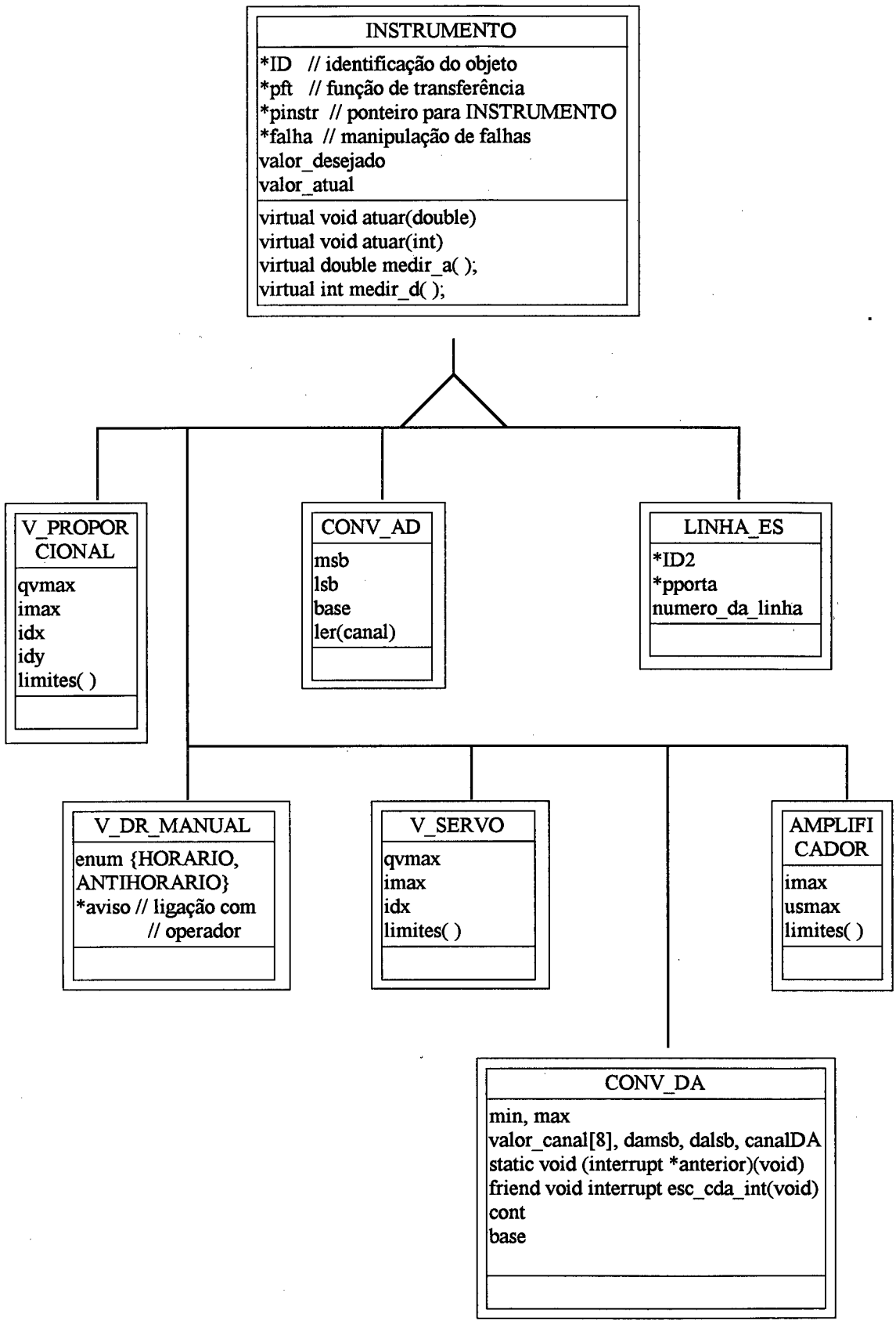


Figura 8.7 - Hierarquia de instrumentos

```
class INSTRUMENTO
{
protected:
 INSTRUMENTO *pinstr; // ponteiro para instrumento
          .
          .
          .

public:
 INSTRUMENTO(char *ID, FT *pfta=NULL, INSTRUMENTO *pinstr=NULL);
          .
          .
          .
};
```

identificação do objeto

função de transferência

ligação com outro instrumento

Figura 8.8 - A classe INSTRUMENTO

A função de transferência é definida fora do escopo da classe instrumento, sendo ligada a esta no momento da instanciação de cada **objeto-imagem**, evitando-se a criação de classes especializadas para cada função de transferência. Este procedimento levou a criação de uma biblioteca de funções de transferência, aproveitando-se também o fato de existirem funções de transferência iguais para instrumentos diferentes.

9 - Conclusões

9.1 - Contribuições

No Desenvolvimento de Software tradicional, como também na Engenharia, os elementos de um sistemas são abstraídos por funções, relegando para um segundo plano os fenômenos ou dados na construção do modelo. Na produção de programas segundo este raciocínio, é comum existir a representação de um componente (como por exemplo uma válvula que recebe uma determinada corrente de pilotagem) com a atenção deslocada para sua função, e não para seu estado em decorrência da grandeza por ele representado (no caso o valor de atuação da válvula). Neste contexto, o elemento atua mais como um componente de simulação, cuja mensagem recebida não carrega o novo valor da grandeza do meio físico na qual o projetista deseja introduzir modificações. Além disto, os outros instrumentos a ele conectados (amplificador, conversor, relés, etc.) adquirem papéis auxiliares e geralmente não são descritos de maneira equivalente no modelo.

O uso do **objeto-imagem** como forma de abstração proporciona a construção de uma solução computacional que reflete bem as disposições e ligações dos elementos reais, aumentando a inteligibilidade do sistema sob modelagem. A concepção da coordenação é substancialmente facilitada, uma vez que os valores das mensagens fornecidas aos subsistemas correspondem efetivamente aos do processo energético-material, aumentando o nível de abstração e independência, e permitindo que o desenvolvimento de subsistemas possa ser feito em paralelo com o gerenciamento central. Além disto são produzidos dois níveis de reutilização, um de subsistemas (SA, SM e SASM) e outro dos componentes destes (**objetos-imagem**). A abstração dos elementos de interfaceamento confere um grau de visualização dos sistemas geralmente não fornecido por técnicas que não dão tratamento uniforme a todos os elementos.

Outro fator positivo é a possibilidade de substituição simples dos elementos reais por elementos de simulação, decorrente da fácil visualização dos limites entre **software/hardware** produzidos. A construção das imagens e dos objetos de simulação podem ocorrer em paralelo, substituindo-se as relações entre entrada/saída e funções de transferências. Uma vez concluída a simulação, efetua-se a troca dos objetos de simulação por elementos físicos, sem que haja necessidade de mudanças na estrutura de coordenação (programa principal)[23].

Em trabalhos mais recentes[24] examina-se o desenvolvimento conjunto de **hardware software** a partir de um modelo apresentado na notação canal/agência. Neste caso o início do trabalho se concentra em um modelo central[25] que fornece uma notação comum para o projeto de **hardware** e **software**. A partir daí os especialistas de cada área tomam decisões quanto ao projeto detalhado e implementação do modelo, empregando metodologias, ferramentas e materiais adequados.

O emprego do **objeto-imagem** pode aumentar a complexidade na codificação, induzida na modelagem de sistemas simples, principalmente aqueles que possuem componentes que não necessitariam ser abstraídos em classes. Este efeito é eliminado através de simplificações (agrupamentos e eliminações de redundâncias), o que também reduz ou elimina a acréscimo de tempo em função das chamadas de métodos adicionais. Experiências mostram que estas simplificações devem de preferência ser realizadas no projeto detalhado e/ou codificação, uma vez que pode existir uma mudança de paradigma (substituição de classes por função), comprometendo a inteligibilidade do sistema e perda de versatilidade na reutilização.

---

23 Neste caso o modelo seria acrescentado de características de simulação adequadas.

24 Trabalho de LÜCKE, DE NEGRI & MADEIRA apresentado na CONINFO em maio de 1995.

25 Como exemplificado pela figura 7.8.

9.2 - Recomendações

O artifício do objeto-imagem permite a descrição estrutural do sistema físico e a própria interação com este. Até o momento foram utilizados nos sistemas contínuos a operação sem realimentação, sendo necessária uma investigação prática para a modelagem de malhas fechadas e sua adequação a teoria de controle. Alguns testes mostraram a aplicabilidade da técnica nestas condições.

Como continuação destre trabalho estariam incluídos a construção de hierarquias de classes para componentes de outras áreas; a construção de um ambiente para simulações; uma ferramenta de especificação gráfica bem como uma formalização na especificação dos sistemas.

Anexo

Código utilizado para testes de eficiência em C++ descrito no capítulo 5

Programa 1

```
// Este programa implementa as chamadas sem argumentos e sem retorno
// MNM 28\03\95  UFSC
//

#include <conio.h>
#include <stdio.h>


#define OUT_HI  outp(0x378,0xFF);
#define OUT_LOW outp(0x378,0);
int cnt,t,ciclo,n_chamadas; // auxiliares

int a,b,c;
void outG(); //função global

// classes nao virtuais (base e derivada)

class CLASSE_NV{
        int a,b,c;
public:
        CLASSE_NV(){a=b=0;};
        void outIL()
        {
                c=a+b;
        };
        void outNV();
};
class CLASSE_DNV:public CLASSE_NV {
        int a,b,c;
public:
   CLASSE_DNV(){a=b=0;};
};


// classes virtuais (base e derivada)

class CLASSE_V{
        int a,b,c;
public:
   CLASSE_V(){a=b=0;};
   virtual void outV();
};

class CLASSE_DV:public CLASSE_V {
        int a,b,c;
public:
   CLASSE_DV(){a=b=0;};
   void outV();     // substitui o metodo da classe base
};
```

```
 CLASSE_NV  tnv;          // objeto de classe nao virtual
   CLASSE_DNV tdnv;         // objeto de classe derivada nao virtual
   CLASSE_V   tv;           // objeto de classe virtual
   CLASSE_DV tdv1;          // objeto de classe derivada virtual

   CLASSE_V   *ptv = new  CLASSE_DV();   // faz o ponteiro apontar para o
                                         // objeto de classe derivada

// programa principal
main()
{

   int sair = 0;
   puts("\n\n   Funcoes sem parametro e sem retorno");
   puts("\n Entre com o numero de ciclos - >");
   scanf("  %d",&ciclo);
   puts("\n\n Entre com o numero de chamadas - > ");
   scanf("  %d",&n_chamadas);

do{
   puts(" \n\nENTRE COM A OPCAO:\n\n0) chamada vazia\n1) chamada global\n2) chamada CBNV
inline\n3) CBNV\n4) CDBNV\n5) CBV\n6) CDBV\n7) PDBV\n8) sai\n\n\n");
   getchar();
   char ca=getchar();
   switch(ca-48)
   {
          case 0:
// chamada vazia
//---------------------------------------
          puts(" opcao 0 escolhida \n");
   for(cnt=0;cnt<ciclo;cnt++)
          {
                 OUT_HI
                 for(t = 0; t < n_chamadas; t++){}
                 OUT_LOW
          }
      break;
      case 1:

// chamada da funcao global
//---------------------------------------
          puts(" opcao 1 escolhida \n");
   for(cnt=0;cnt<ciclo;cnt++)
          {
                 OUT_HI
                 for(t = 0; t < n_chamadas; t++)
                        outG();
                 OUT_LOW
          }
      break;
      case 2:

// chamada da funcao in-line
//---------------------------------------
          puts(" opcao 2 escolhida \n");
   for(cnt=0;cnt<ciclo;cnt++)
   {
                 OUT_HI
                 for(t = 0; t < n_chamadas; t++)
                        tnv.outIL();
                 OUT_LOW
          }
```

```
        break;
        case 3:
// chamada de metodos definidos na classe nao virtual
//-----------------------------------------
                puts(" opcao 3 escolhida\n");
    for(cnt=0;cnt<ciclo;cnt++)
                {
                        OUT_HI
                        for(t = 0; t < n_chamadas; t++)
                                tnv.outNV();
                        OUT_LOW
                }
        break;
        case 4:
                puts(" opcao 4 escolhida\n");
// chamada de metodos definidos na classe base atraves
// da classe derivada
//-----------------------------------------
    for(cnt=0;cnt<ciclo;cnt++)
                {
                        OUT_HI
                        for(t = 0; t < n_chamadas; t++)
                                tdnv.outNV();

                        OUT_LOW
                }
                break;
        case 5:

// chamada de metodos virtuais da classe base virtual
//-----------------------------------------
                puts(" opcao 5 escolhida\n");
    for(cnt=0;cnt<ciclo;cnt++)
                {
                        OUT_HI
                        for(t = 0; t < n_chamadas; t++)
                                tv.outV();
                        OUT_LOW
                }
                break;
        case 6:

// chamada de metodo virtual definido na classe base
// atraves da classe derivada
//-----------------------------------------
                puts(" opcao 6 escolhida\n");
    for(cnt=0;cnt<ciclo;cnt++)
                {
                        OUT_HI
                        for(t = 0; t < n_chamadas; t++)
                                tdv1.outV();
                        OUT_LOW
                }
        break;
        case 7:
// chamada de metodo de classe derivada atraves de
// ponteiro para a classe base
//-----------------------------------------
                puts(" opcao 7 escoihida\n");
    for(cnt=0;cnt<ciclo;cnt++)
                {
                        OUT_HI
```

```
                    for(t = 0; t < n_chamadas; t++)
                        ptv->outV();
                    OUT_LOW
            }
        break;
            case 8:
            sair = 1;
    } // fim switch
}while(sair==0);
  return 0;

}

//---------------------------------------
//
// Implementacoes das funcoes
// ( todas utilizam a macro OUT definida no
//  inicio do programa )
//---------------------------------------
void outG()
{
            c=a+b;
}

void CLASSE_NV::outNV()
{
            c=a+b;
}
void CLASSE_V::outV()
{
            c=a+b;
}
void CLASSE_DV::outV()
{
            c=a+b;
}
```

Programa 2

```
// Este programa implementa as chamadas com um argumento e sem retorno
// MNM 28\03\95  UFSC
//

#include <conio.h>
#include <stdio.h>


#define OUT_HI outp(0x378,255);
#define OUT_LOW outp(0x378,0);
int cnt,t,ciclo,n_chamadas; // auxiliares

int a,b,c;
void outG(int p);

// classes nao virtuais (base e derivada)

class CLASSE_NV{
   int a,b,c;
public:
            CLASSE_NV(){a=b=0;};
```

```
        void outIL(int p)
        {
    c=a+b;
        };
        void outNV(int p);
};
class CLASSE_DNV:public CLASSE_NV {
  int a,b,c;
public:
  CLASSE_DNV(){a=b=0;};
};


// classes virtuais (base e derivada)


class CLASSE_V{
  int a,b,c;
public:
  CLASSE_V(){a=b=0;};
  virtual void outV(int p);
};

class CLASSE_DV:public CLASSE_V {
  int a,b,c;
public:
  CLASSE_DV(){a=b=0;};
  void outV(int p);    // substitui o metodo da classe base
};


int sair = 0;
  CLASSE_NV  tnv;        // objeto de classe nao virtual
  CLASSE_DNV tdnv;       // objeto de classe derivada nao virtual
  CLASSE_V   tv;         // objeto de classe virtual
  CLASSE_DV  tdv1;       // objeto de classe derivada virtual

  CLASSE_V   *ptv = new CLASSE_DV();   // faz o ponteiro apontar para o
                                       // objeto de classe derivada


// programa principal igual ao da listagem do programa 1
```

Programa 3

```
// Program para testes de velocidade entre algumas formas de acesso
// a funções e métodos
// Este programa implementa as chamadas com um argumento e retorno
// MNM 28\03\95  UFSC
//

#include <conio.h>
#include <stdio.h>

#define OUT_HI outp(888,255); // escreve 1 em todas as linhas da interface paralela
#define OUT_LOW outp(888,0);
// escreve 0 em todas as linhas da interface paralela
int cnt,t,ciclo,n_chamadas; // variáveis auxiliares

int a,b,c;          //variáveis globais utilizadas por outG()
int retorno;              //variável que recebe os retornos
```

```
int parametro = 1;          //variável para passagem como parametro
int outG(int p);    //função global

// classes não virtuais (base e derivada)

class CLASSE_NV{
   int a,b,c;
public:
          CLASSE_NV(){a=b=0;};
          int outIL(int p)
          {
      c=a+b;
                    return c;
          };
          int outNV(int p);
};
class CLASSE_DNV:public CLASSE_NV {
   int a,b,c;
public:
   CLASSE_DNV(){a=b=0;};
};


// classes virtuais (base e derivada)


class CLASSE_V{
   int a,b,c;
public:
   CLASSE_V(){a=b=0;};
   virtual int outV(int p);
};

class CLASSE_DV:public CLASSE_V {
   int a,b,c;
public:
   CLASSE_DV(){a=b=0;};
   int outV(int p);     // substitui o método da classe base
};


   CLASSE_NV  tnv;          // objeto de classe nao virtual
   CLASSE_DNV tdnv;         // objeto de classe derivada não virtual
   CLASSE_V   tv;           // objeto de classe virtual
   CLASSE_DV  tdv1;         // objeto de classe derivada virtual

   CLASSE_V   *ptv = new  CLASSE_DV();   // faz o ponteiro apontar para o
                                         // objeto de classe derivada

// programa principal igual ao do programa 1
```

Referências Bibliográficas

BOOCH, Grady. **Software engineering with ada**. Estados Unidos : Benjamin/Cummings Publishing Company, 1983.

_____. **Object oriented design with applications**. Redwood City, USA : Benjamin/Cummings, 1991.

_____. **Designing an application framework**. Dr. Dobbs Journal. US : Miller Freeman Inc. Fevereiro de 1994. CD-ROM.

BURR-BROWN COMPANY. **The handbook of personal computer instrumentation**. Estados Unidos : Intelligent Instrumentation Inc, 1991.

_____. _____. 1994.

CAMPBELL, Roy H.; ISLAN, Nayeem; RAILA, David; MADANY, Peter. Designing and implementing choices: an object-oriented system in C++. **Communications of the ACM,** New York, v.36, n. 9. p.117-126. Sept 1993.

CHEN, P.P.S. The Entity-relationship model - toward a unified view of data. **ACM Transactions on Database Systems 1**, Mar. 1976.

COAD, P., YOURDON, E. **Object-oriented analysis**. Estados Unidos : Prentice Hall, 1991.

DOEBLIN, Ernest O. **Measurement systems - application and design**. 3 ed. Estados Unidos: McGraw-Hill International Book, 1983.

ECKEL, B. **Using C++**. Estados Unidos: Osborn McGraw-Hill, 1989.

EGGBRECHT, LEWIS C. **Interfacing to the IBM Personal Computer**. Estados Unidos : Howard W. Sams & Co., 1987.

ELLIS, John R. **Objectifying real-time systems.** Estados Unidos : SIGS Books, 1994.

ELLIS, A. Margaret; STROUSTRUP, Bjarne. **C++ manual de referência comentado,** Brasil: Editora Campus, 1993

EMBLEY, David W., BARRY D. Kurtz, WOODFIELD, Scott N. **Object-oriented systems analysis: a model-drivel approach**. New Jersey : Prentice-Hall, 1992.

FIEDLER, J., RIX, F., ZÖLLER, H. **Objekt-Orientierte programmierung in der automatisierung.** Düsseldorf, Germany: VDI-Verlag, 1991.

FREDERICK, D. K., CARLSON, A. B. **Linear systems in communication and control**. Estados Unidos : John Wiley & Sons, 1971.

GILBERT, Richard. The General-Purpose Interface Bus. **IEEE Micro**, p41-51, Feb. 1982.

HANISH, H.M. **Petri-Netz in der Verfahrenstechnick.** München: Oldenburg, 1992.

HENDERSON-SELLERS, B., EDWARDS, J. M. Oriented Systems Life Cycle. **Communications of the ACM,** New York, V.33, p.143-159, Sept. 1990.

HEUSER, C. A. **Modelagem conceituai de sistemas**. In: V ESCOLA BRASILEIRO-ARGENTINA DE INFORMÁTICA. 1990.

HEWLETT-PACKARD. **Interfacing concepts,** [S.l : s.n.], 1983

HORDESKI, Michael F. **Microprocessors in industry.** Estados Unidos : Van Nostrand Reinhold Company Inc. 1984.

INTEL CORPORATION. **Microcomunications handbook**. Estados Unidos : [n.s.], 1989.

JACOBSON, I., MAGNUS, C., JONSSON, P., ÖVERGAARD, G. **Object-oriented software engineering**. Estados Unidos : Addison-Wesley Publishing Company, 1992.

JONES, Barry E. **Instrumentation, measurement and feedback**. UK : McGraw-Hill Book Company, 1977.

JORDAN, David. Implementation benefits of C++ language mechanisms. **Communications of the ACM**, New York, v. 33, n. 9, p. 61-64, Sept. 1990.

KARAORMAN, Murat; BRUNO, John. Introducing Concurrency to a Sequential Language. **Communications of the ACM**, New York, v. 36, n. 9., p. 103-116, Sept. 1993.

LISKOV, Barbara; GUTTAG, John. **Abstraction and Specification in Program Development**. Estados Unidos : Mcgraw-Hill, 1986.

LÜCKE, H. A. H., DE NEGRI, V. J. **Uma metodologia integrada para o desenvolvimento de sistemas automáticos. In: 2 CONGRESO INTERAMERICANO DE COMPUTACIÓN APLICADA A LA INDUSTRIA DE PROCESOS**, 1994, Santiago - Chile. *Anais...* Santiago: CIT, 1994. 290 p.6-13.

LÜCKE, H. A. H., DE NEGRI, V. J., MADEIRA, M.N. Uma metodologia para a programação orientada a objetos na automação industrial. **Revista Máquinas e Metais**, São Paulo: Aranda, n. 353 p.108-115, jun. 1995.

MEYER, Bertrand. Reusability: the case for object-oriented design. **IEEE Software**, p.50-64, Mar. 1987.

MEYER, Bertrand. Lessons from the design of the Eiffel libraries. **Communications of the ACM**, New York, v. 33, n. 9, Sept 1990.

MICROSOFT CORPORATION. **Programmer's guides - Visual C++**. Estados Unidos: Microsoft Press, 1993.

MINOURA, T., PARGAONKAR S.S., REHFUSS K. Structural active object systems for simulation. **OOPSLA'93**, p. 338-355, 1993.

MOOG. **Direct drive servo valve with integrated electronics**. Alemenha, 1994. Nota de Catálogo.

NATIONAL INSTRUMENTS. **IEEE 488 and VXIbus control, data acquisition, and analysis**. Estados Unidos: [n.s.], 1993.

NEC. **mPD7210 Intelligent GPIB interface controller**. Japão, 1983. Nota de Catálogo.

NERSON, Jean-Marc. Applying object-oriented analysis and design. **Communications of the ACM**. Volume 35, número 9, p.63 - 74, Sept. 1992.

OMRON. **E2E-X2**. Brasil, 1994. Nota de Catálogo.

OSBORN, Adan; **An introduction to microcomputers: volume 1**, 2.ed. Estados Unidos: Osborne/McGraw-Hill, 1980.

PARNAS, D. On the Criteria to be Used on Decomposing Systems Into Modules. **Communications of the ACM**. Volume 15, 12, p.1053-1058. 1972.

PASCOE, G. August. Elements of object-oriented programming. **Revista Byte**, Estados Unidos, v. 11. Aug. 1986.

ROBERT BOSCH GMBH. **Electrohydraulic proportional valves and closed loop control Valves: theory and application**. Alemanha : [n.s.], 1989.

____. **Proportional Control Valves**. Alemanha, 1992,. Nota de catálogo.

RUMBAUGH, James; BLAHA, Michel; PREMERLANI, Willian et al. **Object-oriented modelling and design**, New Jersey : Prentice-Hall, 1991.

SCHNEIDER, CARLOS; LÜCKE, H. A. H. **Sistemas de medição**. Apostila do Curso de Engenharia Mecânica, Universidade Federal de Santa Catarina, 1989.

SELIC, Bran; GULLEKSON, Garth; WARD, Paul D. **Real-time object-oriented modeling**. United States of America : John Wiley & Sons, Inc., 1994.

SMITH, L.S.P.S., GAWTHROP, P.J. **An Environment for industrial process design using Bond Graph modelling.** In: BOND GRAPHS FOR ENGINEERS, 1992, North-Holand: P. C. Breedveld, 332 p. 291-300.

SMITH, Ernest. **Principles of industrial measurement for control applications**. Estados Unidos : Instrument Society of America, 1984.

STEIGLITZ, KENNETH. **An introduction to discrete systems,** US : John Wiley & Sons, Inc., 1974.

TITUS, Jonathan A.; TITUS, Peter Christopher A.; RONY, Peter R. **Microcomputer - Analog Converter Software & hardware Interfacing**, Estados Unidos : Howard W. Sams & Co., Inc., 1979.

WECK, MANFRED. **Automatisierung und steurungstechnik**. Alemanha: VDI-Verlag GmbH, 1989.

WEGNER, Peter. Learning the language. **Revista Byte**, p.245-253, Mar. 1989.