

Apostila de Automação Industrial

João Paulo Cerquinho Cajueiro

26 de fevereiro de 2016

Sumário

1 Sistemas de automação	3
1.1 História	4
1.2 Classificação	5
1.3 Pirâmide de automação	6
2 Instrumentação Industrial	8
2.1 Sensores discretos	9
2.1.1 Sensores de contato	9
2.1.2 Sensores de proximidade	11
2.1.3 Chaves de processo	14
2.2 Sensores Contínuos	15
2.3 Transmissão de dados, aterramento e blindagem em instrumentação.	15
3 Controladores Lógico-Programáveis (CLP)	16
3.1 Dispositivos Eletromecânicos	17
3.2 De relês a CLPs	18
3.2.1 Diagrama Ladder	19
3.3 IEC61131	22
4 Programação - arduino	25
4.1 Piscando um led.	26
4.2 Sinais analógicos	28
4.3 Controle: for e if	29
4.4 Comunicação com o computador	30
4.5 Ladder como uma linguagem	32
4.6 Memória	33
4.6.1 Subida e descida de uma entrada.	34
4.7 Contadores	37
4.7.1 Acesso às variáveis	37
4.8 Temporizadores (timers)	39
4.9 plcLib	40

4.9.1	Acumulador	41
4.9.2	Funções lógicas	42
4.9.3	Memória	44
4.9.4	Temporizadores	45
4.9.5	Contadores	46
5	Linguagem grafset	48
5.1	Divergências e convergências	49
5.2	Ações	50
5.3	Níveis	53
6	SCADA – Supervisory Control and Data Acquisition	58
6.1	Arquitetura de sistemas SCADA e interfaceamento com níveis de automação	58
6.2	Funcionalidades principais de sistemas SCADA	58
7	Sistema SCADA MANGO	59
7.1	Instalação	59
7.2	Uso	59
7.3	<i>Watch list</i>	61
7.4	Tela gráfica	61
8	Gerenciamento da manufatura: PIMS e MES	67
8.1	PIMS	67
8.1.1	Historiador de Processos	68
8.1.2	Banco de Dados Temporal	70
8.1.3	Interface Gráfica	70
8.2	MES	71
8.2.1	Redes industriais	75
8.3	Gerenciamento da manufatura: MES	78
8.4	Redes de Comunicação: Introdução e noções básicas	79
8.4.1	Modelo OSI	80
8.4.2	Internet	83

Capítulo 1

Sistemas de automação

A palavra automação vem do latim *automatus* – mover por si mesmo. Logo a automação de uma tarefa consiste em fazer esta tarefa ser realizada sem trabalho humano. Isto pode ser por diversos motivos: seja por que é uma tarefa perigosa e portanto queremos aumentar a segurança das pessoas, como num processo que envolva alta temperatura, por exemplo; seja para fazer a tarefa de forma mais rápida, seja para melhorar a qualidade do produto final ou seja porque simplesmente o custo do trabalho humano é muito elevado. Logo, podemos definir automação da seguinte forma:

Automação é a substituição do trabalho humano para melhorar segurança, qualidade, produção e custos.

Neste contexto, automação industrial é nada mais que a automação de um sistema industrial, ou de um sistema de manufatura. Embora manufatura venha de fazer com as mãos, a revolução industrial mudou seu conceito, passando a significar a fabricação de praticamente qualquer produto. Do ponto de vista econômico, a manufatura é a transformação de materiais (matéria prima) em itens de maior valor (produto). Isto é conseguido por uma determinada sequência de processos químicos e físicos. De forma mais sucinta:

Manufatura é a transformação de matéria prima em produtos pela aplicação de um ou mais processos.

Logo, a automação industrial consiste em fazer os processos necessários para a manufatura com o mínimo de esforço ou interferência humana, visando melhor segurança, qualidade, produção e custo. Note que por esforço humano, queremos dizer tanto esforço físico quanto mental, logo uma máquina bastante complexa mas que funcione a manivela, não se classificaria como automação; da mesma forma uma máquina que não exija esforço físico mas requer atenção constante também não é automatizada (seria apenas mecanizada).

1.1 História

É interessante pegar alguns pontos chaves na história da automação. Embora várias máquinas mecânicas de diversas graus de complexidade já existissem, como por exemplo relógios mecânicos desde o século VIII na China e desde o século XIII na Europa e os fantásticos robôs de Pierre Jaquet-Droz, do século XVIII, considera-se que a revolução industrial iniciou com a invenção do tear mecânico por Cartwright em 1785, que realiza um movimento relativamente complexo de forma automática a partir de uma roda d'água.

Um outro grande avanço ocorreu por volta de 1788, com a invenção do mecanismo de regulagem de fluxo de vapor de James Watt, o que permitia então controlar a potência de caldeiras e outras máquinas a vapor, controlando uma energia

muito maior que uma roda d'água. Isto foi um grande impulso para a mecanização, mas a verdadeira automatização ainda ficava muito restrita devido a dificuldade de realizar processos complexos de forma automática. Ou seja, retirava-se grande parte do esforço físico do homem, mas ainda era necessário muito esforço mental.

Em 1820 Babbage começou a desenvolver a sua máquina diferencial, que hoje chamaríamos de uma calculadora mecânica. Ela evoluiu até o conceito da máquina analítica, descrita em 1837, que é considerada o primeiro projeto de computador, embora apenas partes dela tenham sido efetivamente construídas.

Em 1880, Herman Hollerith criou um novo método baseado na utilização de cartões perfurados, para automatizar algumas tarefas de tabulação do censo dos EUA que antes duravam 10 anos. Com o método, o processo era concluído em 6.

Ao longo da primeira metade do século XX foram utilizados muitos sistemas eletromecânicos para o controle de processos industriais. Eram os chamados circuitos chaveados, que utilizavam relés para o controle lógico e para o comando de motores.

Em 1936, Alan Turing descreveu um *computador universal* em seu artigo “On Computable Numbers, with an Application to the Entscheidungsproblem”, o que hoje é conhecido como uma Máquina de Turing. Suas idéias foram desenvolvidas em 1944, com a construção do Colossus, considerado como o primeiro computador, embora tivesse a função específica de quebrar o código criptográfico alemão na segunda guerra. Ele consistia de um circuito com 1600-2400 válvulas com capacidade de processamento de 25k caracteres/s. A título de comparação, os computadores de casa de hoje em dia atingem 10 bilhões de cálculos por segundo.

O avanço da eletrônica fez que a capacidade de processamento dos computadores aumentasse de forma exponencial. Atualmente o mais rápido computador do mundo é o chinês Tianhe-2, que faz 33,86 quatrilhões de cálculos por segundo, consumindo 24MW.

Na década de 60 a General Motors fez uma especificação de um CLP – Controlador Lógico Programável, que é um computador voltado para o controle de processos industriais. Em 1968 foi criado o primeiro.

Hoje em dia toda automação está relacionada a sistemas computadorizados, seja em CLPs, CNC, robôs industriais, automação dos sistemas de apoio a produção, entre outros.

1.2 Classificação

Hoje em dia se definem, grosso modo, 3 tipos de automação, tal qual mostra a figura 1.1: fixa, flexível e programável.

A automação fixa é aplicada à produção de um único produto (ou com mínimas variações), em grandes quantidades: refinaria de petróleo, parafuso, tampas de

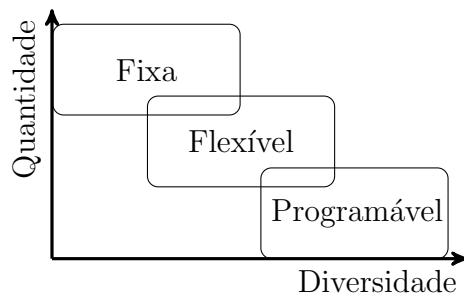


Figura 1.1: Tipos de automação industrial quanto a quantidade e diversidade de produtos.

garrafa, clips, biscoito, cerveja, etc. Ela utiliza equipamentos específicos para aquela tarefa, que portanto tem alto custo mas grande produtividade.

A automação flexível é aplicada à produção de produtos parecidos, em que pequenas modificações permitem a alteração do produto, como por exemplo mudança de um perfil a ser prensado ou extrudado ou a mudança das quantidades do mesmo conjunto de matérias primas (mudança de receita). Tipicamente é feita a chamada fabricação em lotes, onde entre um lote e outro se alteram as peças e/ou as sequências a serem seguidas de forma automática para ter o menor tempo parado possível. Exemplos são livros, circuitos integrados, potes de plástico, máquinas de café.

A automação programável é para produção de produtos diferentes mas cujo volume de produção não justifica um processo único. Ela usa máquinas de propósito geral, tais como robôs, ferramentas de controle numérico e impressoras 3d, onde a definição do processo é quase toda feita por *software*, de modo que o custo do maquinário é diluído em diversos produtos.

A tendência é cada vez mais ter a automação flexível e programável aumentando a capacidade de produção, de modo que a flexível vai ocupando nichos da fixa e a programável da flexível. Apesar disso, em vários casos é difícil imaginar alguns produtos deixando de utilizar a automação fixa.

1.3 Pirâmide de automação

A automação em larga escala de uma grande indústria ou de um conjunto de indústrias é mais complexa que a manufatura: envolve problemas de abastecimento, armazenagem, análise de mercado, exigências ambientais, entre várias outras coisas. Uma forma de se separar os diferentes problemas da automação é através da chamada Pirâmide de Automação, mostrada na figura 1.2.

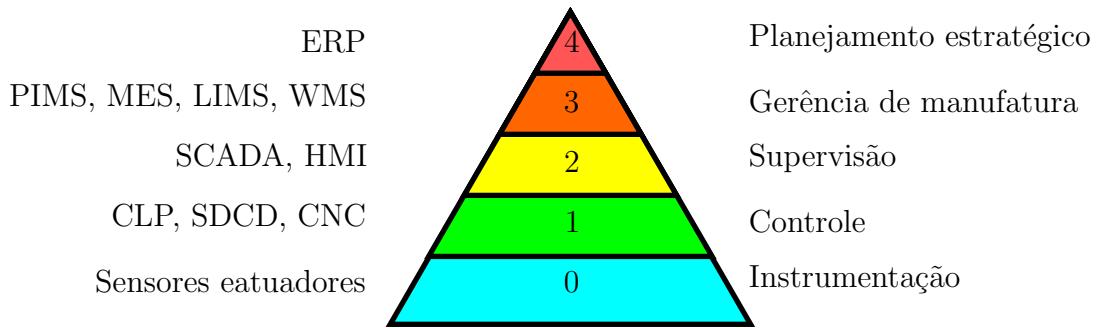


Figura 1.2: Pirâmide de automação.

Note que esta não é a única representação da pirâmide: algumas começam pelo 1, outras têm apenas 4 camadas, e assim por diante, logo mais importante que o número de cada camada é o que tais camadas significam.

Nível 0 :Instrumentação Camada onde se encontram instrumentos, sensores, motores, máquinas, etc. Consistem nos equipamentos do chamado “*chão de fábrica*”.

Nível 1: Controladores Controle automático da planta – onde se localizam os Controladores Lógico-Programáveis (CLP), os Sistemas Digitais de Controle Distribuído (SDCD), os Controles Numéricos Computadorizados (CNC) e/ou computadores de controle.

Nível 2: Supervisão Supervisão e controle do processo através de Interfaces Homem-Máquina (IHMs) ou SCADA (*Supervisory Control And Data Acquisition*).

Nível 3: Gerenciamento da Manufatura Gestão dos recursos da planta e controle da produção. Sistemas PIMS (*Process Information Management System*) e MES (*Manufacturing Execution Systems*).

Nível 4: Gerenciamento da Empresa Gestão dos recursos e produção da empresa como um todo. ERP – *Enterprise Resources Planning*.

Este texto faz um estudo da automação industrial de modo *top-down*: começando do nível 3 até o nível 0. O nível 4 é mais importante para um estudo de engenharia de processo e portanto não será abordado.

Capítulo 2

Instrumentação Industrial

A instrumentação é relativa aos equipamentos que obtém informações acerca do estado de um determinado processo industrial. Os chamados sensores e transdutores.

A definição de sensor e transdutor está longe de ser uma unanimidade. Do ponto de vista da instrumentação, define-se:

Sensor é um elemento que gera um sinal (normalmente elétrico) a partir de uma grandeza física (calor, luz, som pressão etc).

Transdutor é um dispositivo que converte um sinal de uma grandeza para outra.

De onde se tira que todo sensor é um transdutor, mas não o contrário. Em geral os sensores se utilizam de transdutores para converter uma grandeza específica para outra mais facilmente manipulável.

Outra classificação normalmente usada é a de sensores passivos ou ativos:

Sensores ativos geram um sinal de saída sem a necessidade de alimentação externa. Exemplos: termopar, célula fotoelétrica.

Sensores passivos requerem uma entrada de energia para gerar um sinal de saída. Exemplos: Termoresistência, sensor capacitivo.

A maioria dos sensores industriais são passivos.

Outra classificação bastante útil é a de sensores discretos ou contínuos:

Sensores discretos geram uma saída discreta, normalmente binária – do ponto de vista da saída elétrica agem como chaves e são comumente chamados de sensores.

Sensores contínuos geram uma saída que varia continuamente em função da entrada. Pode ser composto por um único transdutor (um resistor por exemplo).

Um complicador é que alguns textos técnicos apresentam uma confusão de sensores e transdutores com sensores discretos (chamados simplesmente de sensores) ou contínuos (chamados erroneamente de transdutores). Portanto deve-se tomar cuidado com o significado destes termos.

2.1 Sensores discretos

Como citado, a maioria dos sensores discretos industriais são chaves elétricas. Neste caso diferenciam-se os sensores de contato, que são chaves eletromecânicas; os sensores de proximidade, que detectam a presença de algum objeto sem tocá-lo; e as chaves de processo.

2.1.1 Sensores de contato

Do ponto de vista da instrumentação, qualquer chave presente no nível 1 da pirâmide que gera sinais lidos no nível 2 realizam logicamente o mesmo tipo de função. Daí que se consideram como sensores de contato:

Botoeiras ou botões, acionados pelo operador do processo.

Chaves de fim de curso que são acionadas mecanicamente por algo no processo.

As botoeiras podem ter diversos formatos e funções, vide a figura 2.1. Podem ser simples botões acionados apenas quando pressionados, interruptores de 3 estados, entre outros. Alguns botões de emergência são acionados quando apertados e só são desligados com o uso de uma chave. Também é comum botoeiras que energizam equipamentos poderem ser travadas na posição desligada com um cadeado. Isto permite que o manutentor trave o equipamento desenergizado enquanto efetua algum serviço.

Chaves de fim de curso são chaves eletromecânicas feitas para serem acionadas por algum produto ou equipamento. São usadas para detectar, por exemplo, a passagem do material sendo processado por um determinado ponto, a posição final de movimentação de algum equipamento, entre outros. Normalmente é implementado como um botão com uma alavanca ou algum outro mecanismo na parte a ser acionada.

As chaves eletromecânicas são relativamente baratas, de uma tecnologia já bem amadurecida e são imunes a interferências eletromagnéticas, porém o movimento a que são submetidas gera desgaste, o que faz com que sua vida útil seja reduzida. Além disso, necessitam do contato com o alvo para ser acionadas, o que pode ser inviável em alguns casos, e tem um tempo de resposta da ordem de milisegundos, que pode ser lento demais para algumas aplicações.



Figura 2.1: Diversos tipos de botoeiras.



Figura 2.2: Exemplos de chaves de fim de curso.

FALTA ESTA FIGURA!

Figura 2.3: Encapsulamentos comuns para sensores de proximidade.

2.1.2 Sensores de proximidade

Existem diversos princípios físicos que podem e são usados para detectar a proximidade de algum material. Estes são chamados de sensores de proximidade e normalmente são aplicados quando se tem algum impedimento ao uso de chaves eletromecânicas.

Para vários casos, é comum o uso de um encapsulamento em formato de rosca, como visto na figura . Isto faz com que vários sensores de proximidade se pareçam, mesmo que usem princípios físicos diferentes.

Indutivo

O sensor indutivo conta com uma bobina na sua extremidade sensora, que gera um campo magnético variável na sua frente. A frequência deste campo magnético depende da própria indutância desta bobina, que por sua vez depende do que estiver na frente do sensor.

Um alvo que esteja na frente deste sensor pode alterar esta indutância por 2 efeitos: se for um condutor, gerará um campo magnético contrário, aumentando a indutância; se for ferroelétrico, concentrará o campo magnético, também aumentando a indutância. Este segundo efeito é maior que o primeiro, o que faz com que este sensor responda melhor a ferro do que a cobre, mesmo com o cobre sendo melhor condutor que o ferro.

A distância que um sensor indutivo consegue detectar um alvo de ferro na sua frente é chamada de **distância sensora nominal**. Para outros materiais esta distância diminui e para materiais não condutores e sem propriedades magnéticas ela cai a zero. A figura 2.4 mostra a variação da distância sensora de sensores indutivos e capacitivos para diferentes materiais.

Além da limitação da distância que pode ser usado e da limitação do material do alvo, sensores indutivos estão sujeitos a interferências eletromagnéticas, que podem gerar falsas detecções.

Capacitivo

Sensores capacitivos usam um circuito oscilador (basicamente o mesmo de sensores indutivos) para detectar a variação da capacitação entre duas placas metálicas na sua ponta. Esta capacitação varia se o alvo tiver uma constante dielétrica diferente da do ar. Funciona muito bem para água e cobre (que tem uma

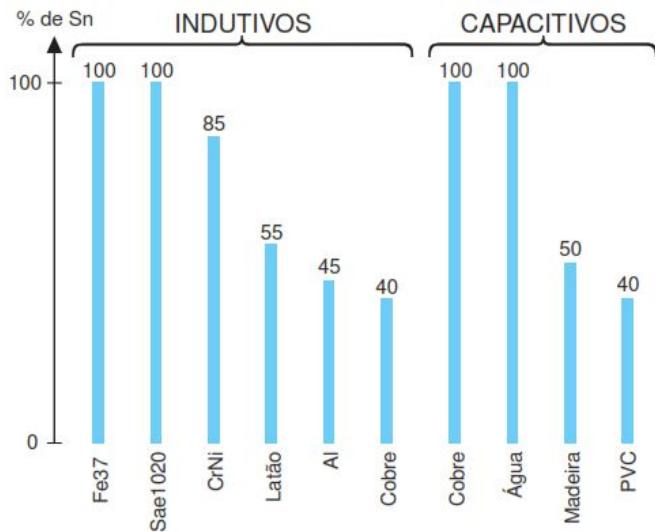


Figura 2.4: Variação da distância sensora de sensores indutivos e capacitivos para diferentes tipos de material.

constante dielétrica 80 vezes maior que o ar). Praticamente não responde para ferro aço e alumínio.

Pela fraca sensibilidade a papel e plástico, pode detectar a presença de alguns objetos dentro da embalagem. Pela alta sensibilidade à água, é muito usado para detecção de nível. Também é sensível à interferência eletromagnética, embora menos que o indutivo.

Ultrassônico

Sensores ultrassônicos detectam a presença de um objeto pelo eco de um sinal ultrassônico (da ordem de 40 kHz). Também é muito usado para sensores contínuos de distância, já que o tempo que o eco demora pode ser usado para determinar a distância até o alvo.

Sensores ultrassônicos podem ser usados para detectar alvos em distâncias de até alguns metros, muito embora aí deva se ter o cuidado de que não há outros elementos que possam gerar eco. É sensível não apenas à qualidade do material mas também à geometria do mesmo.

Outro problema em sensores ultrassônicos é que eles tem uma distância mínima de trabalho. Se o alvo estiver muito próximo o sensor não consegue diferenciar o eco do sinal que ele ainda está gerando.

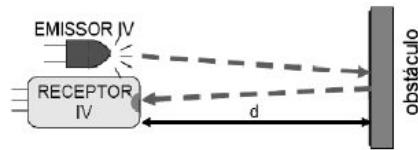


Figura 2.5: Princípio de funcionamento do sensor óptico por refexão.

Óptico

Sensores ópticos funcionam com um emissor e um receptor de luz. Tipicamente se usa luz infravermelha, pois a eletrônica baseada em silício é mais sensível a este comprimento de onda, o que barateia o custo. Normalmente o sinal luminoso é modulado num trem de pulsos, para diminuir a interferência do sol, cuja luz não é modulada.

Os tipos mais comuns deste tipo de sensor são:

Reflexão difusa com o emissor do lado do receptor e detecta-se o alvo pelo seu reflexo.

Retro reflexivo que é o mesmo caso mas com um anteparo reflexivo atrás. Neste caso o alvo impede a passagem da luz.

Barreira que usa o emissor e receptor separados e detecta-se a oclusão do feixe óptico. O uso de um laser como emissor permite alcançar uma grande distância.

Um uso interessante do sensor óptico de barreira é a chamada barreira laser, onde um conjunto de lasers passando por espelhos cercam um equipamento mais perigoso, de modo que qualquer pessoa ou coisa que acione a barreira enquanto o equipamento está atuando cause uma parada de emergência, diminuindo o risco do equipamento.

Além da luz ser imune a influência eletromagnética, o uso de fibras ópticas pode separar bastante a eletrônica do processo, permitindo o uso deste tipo de sensor em regiões onde não pode ter equipamentos elétricos.

Magnético

Há dois tipos básicos de sensores magnéticos: sensores Hall e *reed switches*. Ambos detectam a presença de um campo magnético, normalmente causado pela aproximação de um imã.

O sensor Hall detecta o campo magnético pela influência do mesmo numa corrente elétrica, pelo chamado efeito Hall. É muito usado para medir a rotação

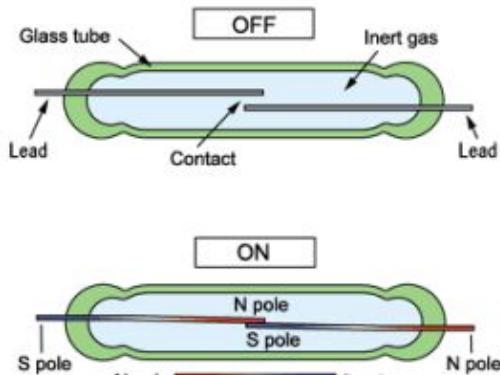


Figura 2.6: *Reed Switch*.

em eixos com um imã acoplado ou em motores elétricos. É um sensor barato, de alta durabilidade e rápido tempo de resposta, porém apenas aplicável a alvos magnetizados.

O *reed switch* é composto de dois contatos de material ferromagnético normalmente separados, tal como mostra a figura 2.6. Um campo magnético perpendicular a estes contatos causam a magnetização dos mesmos, o que faz com que eles se atraiam e fechem contato.

O *reed switch* é bem mais barato que o Hall, mas tem pouca durabilidade e elevado tempo de resposta.

2.1.3 Chaves de processo

Chaves de processo são chaves elétricas que atuam quando uma grandeza do processo de fabricação (uma variável de processo) passa determinado nível. Por exemplo: temperatura do tanque 1 acima de 100 °C, nível do silo abaixo de 2 m, e assim por diante.

São exemplos de chaves de processo:

bóias que acionam um contato elétrico,

sensor capacitivo acionado pelo nível de água de um reservatório,

chaves de fluxo onde uma lingueta aciona uma chave eletromecânica se o fluxo passar de um determinado limite,

termostatos que acionam quando a temperatura passa de determinado nível, entre outros.

É cada vez mais comum trocar as chaves de processo por sensores contínuos e implementar o limite de chaveamento no nível de controle da pirâmide. Isto permite uma maior flexibilidade, pois variar o limite no software é mais simples.

2.2 Sensores Contínuos

2.3 Transmissão de dados, aterramento e blindagem em instrumentação.

Capítulo 3

Controladores Lógico-Programáveis (CLP)

3.1 Dispositivos Eletromecânicos

Há vários dispositivos eletromecânicos de interesse para a automação industrial, dos quais destacamos motores elétricos, solenóides, chaves e relês.

Destes, motores e solenóides são atuadores e chaves são sensores. Relês podem ser usados tanto como atuadores como sensores, embora sejam mais usados em atuadores.

Chaves já foram vistas junto com instrumentação, enquanto motores são assuntos de outras disciplinas e fogem do escopo desta disciplina.

Solenóides são basicamente eletroimãs com um eixo ferromagnético preso a uma mola. O eixo pode ter duas posições: uma mantida pela mola, que é a posição que ele assume quando o eletroimã não é acionado, e outra forçando a mola, quando o eletroimã é acionado. Ou seja: são dispositivos de acionamento binários.

Solenóides tipicamente fornecem uma movimentação de no máximo poucos centímetros a cargas de miligramas a dezenas de quilogramas. São muito usados para acionamento de válvulas - válvulas solenóides.

Relês eletromecânicos são, por construção, solenóides no qual o eixo faz abrir ou fechar contatos elétricos. Eles são chaves elétricas acionadas eletricamente. Hoje em dia existem relês baseados em eletrônica, chamados de relês de estado sólido, que tem a mesma função mas por princípios de funcionamento bem diferentes. Eletricamente pode-se visualizar um relê apenas como um indutor (a bobina do eletroimã) e um contato, e muitas vezes é desta forma que ele é representado em um diagrama esquemático (vide exemplo na Figura 3.1b).

O estado de repouso dos cantatos define o tipo básico do relê: normalmente aberto ou normalmente fechado. É comum relês que tenham 2 ou mais contatos acionadas por uma mesma bobina, inclusive podendo ser uma normalmente aberta e outra normalmente fechada, tal como o relê da figura 3.1a.

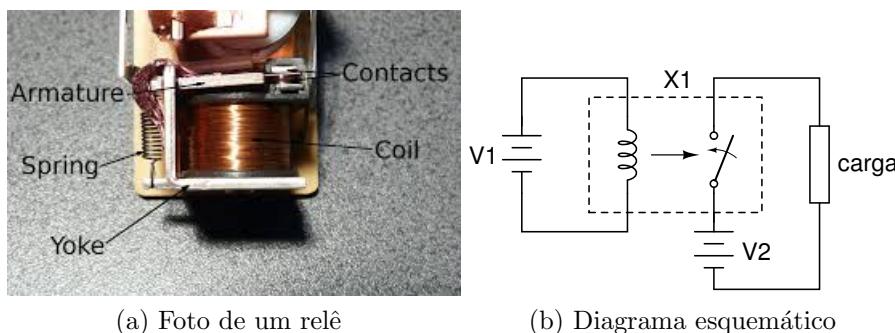


Figura 3.1: Foto de um relê e representação esquemática.

Uma grande utilidade dos relês vem do fato que a bobina de acionamento é eletricamente isolada dos cantatos, o que permite usar relês para:

- Controle de alta tensão e/ou potência através de baixa tensão e/ou potência
– um sinal de mW pode comandar kW de potência através de um relê.
- Isolação e proteção do circuito de controle.
- Acionamento trifásico – um contator é um relê com três conjuntos isolados de contatos acionados pela mesma bobina.
- Implementação de lógica de controle – são os chamados circuitos chaveados.

3.2 De relês a CLPs

Um exemplo de aplicação de relê é mostrado na figura 3.2, onde 2 relês c1 e c2 são usados para o acionamento de um motor trifásico comandado pelas botoeiras b0, b1 e b2.

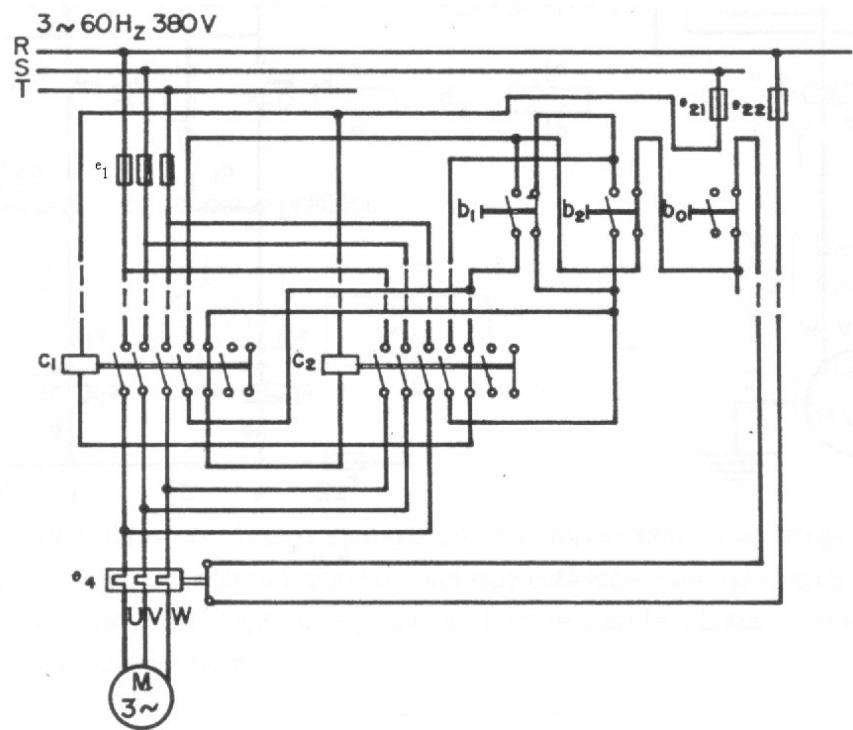


Figura 3.2: Circuito de acionamento de um motor trifásico por relê.

Em geral é mais fácil analisar um circuito do ponto de vista lógico separando a parte de acionamento da parte de controle, então é comum que o símbolo de um relê seja separado em duas partes: a bobina (o eletroímã) e o contato (a chave),

interligados pelo mesmo nome. Isto é exemplificado na figura 3.3, que redesenha o mesmo circuito da figura 3.2 separando a parte de controle da de acionamento, tornando o circuito bem menos convoluto. A ligação entre os diversos contatos e bobinas dos relês permite a realização de diversas funções interessantes para o controle de circuito. Tais circuitos ficaram conhecidos por *circuitos chaveados*.

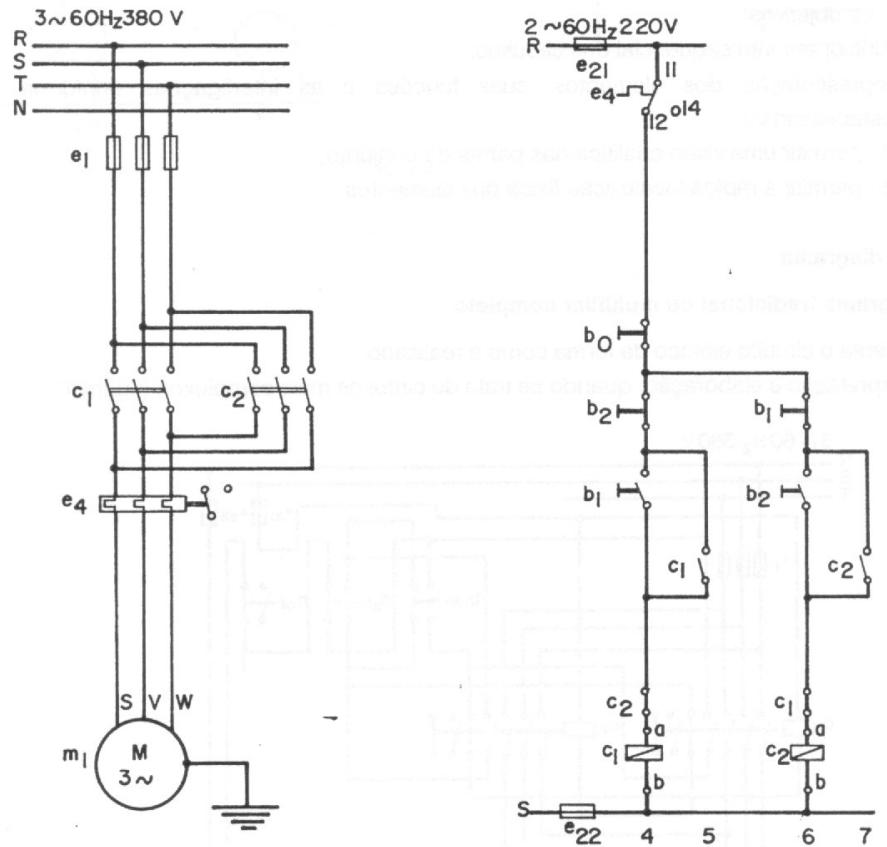


Figura 3.3: Mesmo circuito da figura 3.2, separando a parte de controle da de acionamento.

3.2.1 Diagrama Ladder

Os diagramas ladder são muito utilizados para representar circuitos com relês enfatizando a lógica da ligação. Neste tipo de diagrama as bobinas tem o símbolo $\text{---} \bigcirc \text{---}$, os contatos normalmente abertos são simbolizados por $\text{---} \mid \text{---}$ e os normalmente fechados por $\text{---} \times \text{---}$.

A Figura 3.4 mostra o mesmo circuito de controle da Figura 3.3, só que agora

descrito em ladder. Com os circuitos conectados desta maneira, o diagrama fica parecendo uma escada; daí o nome¹.

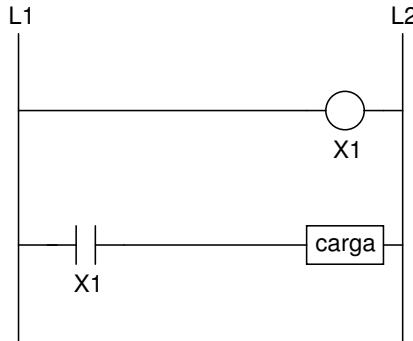


Figura 3.4: Exemplo de circuito de controle em ladder.

Logo de cara nota-se que não existem neste diagrama as tensões V1 e V2. Isto se dá pois neste diagrama considera-se que as tensões estão entre as duas barras L1 e L2 e abstrai-se a fonte de tensão. Isto lembra, de certo modo, a ligação real, com os elementos conectados entre os cabos de fase e o neutro.

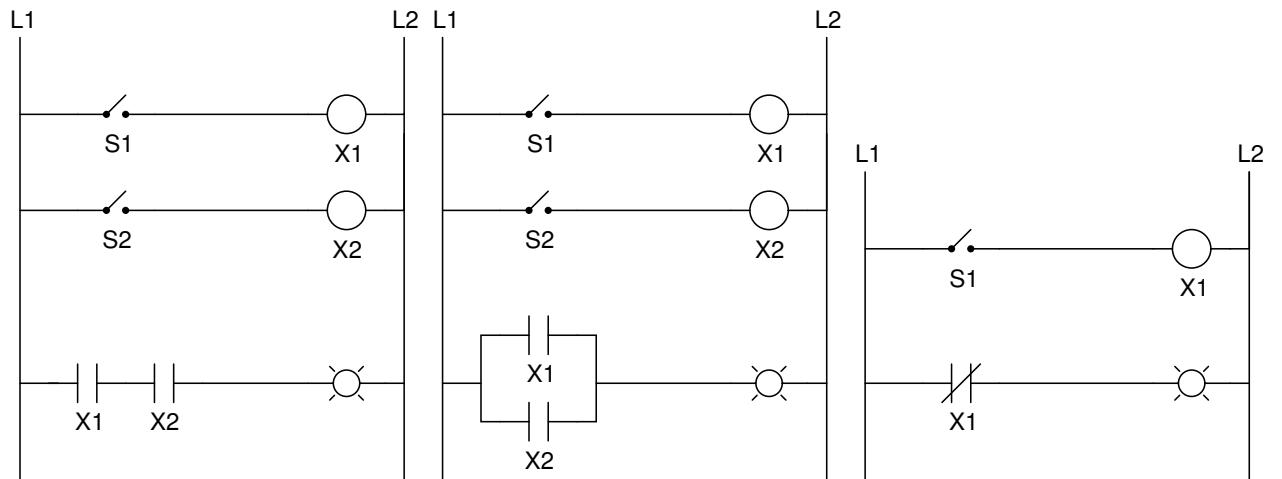
Como exemplo, a figura 3.5 mostra diagramas ladder que acendem uma lâmpada apenas quando 2 relês são acionados, se qualquer um dos relês for acionado ou quando nenhum dos relês é acionado.

O engenheiro Claude Shannon descobriu em 1934 a relação entre os circuitos chaveados e a álgebra de Boole, que é um mapeamento da lógica em uma álgebra. Uma ligação em série realiza um AND lógico (uma multiplicação booleana), uma ligação paralela realiza um OR lógico (uma soma na álgebra de Boole) e o uso de um contato normalmente fechado realiza a inversão lógica, ou o NOT (representado por uma barra sobre a variável). A álgebra de Boole mostra que a combinação destas três operações, e portanto a combinação destes três circuitos, permite realizar qualquer condição lógica para o acionamento do que quer que seja, muito embora a aplicação direta dos postulados e teoremas desta álgebra não seja muito intuitiva.

Porém, apesar da facilidade deste diagrama, a montagem física ainda era bastante complexa, uma vez que o acionamento e os contatos de um relê são obrigatoriamente parte de um único dispositivo físico, e que para lógicas mais complexas essa fiação se tornava bastante complicada. Isto é ainda mais preocupante quando se precisa alterar a lógica de funcionamento, pois se tem que mexer no circuito como um todo.

Além disso, por ser um dispositivo eletromecânico, relês não tem um elevado tempo de vida: pode chavear talvez menos que 10 000 vezes, dependendo da tensão e corrente que suporta.

¹em inglês *ladder* significa escada.



(a) Aciona apenas se ambos relês estiverem acionados.
 (b) Aciona se um ou outro relê estiver acionado.
 (c) Aciona se o relê não estiver acionado.

Figura 3.5: Exemplos de circuitos lógicos chaveados em ladder.

Em 1947, foi inventado o transistor, que age, como o relê, como uma chave elétrica controlada eletricamente. Principalmente pelo fato de não ter partes móveis, um transistor tem um tempo de vida muito mais longo que um relê, além de ter um chavemaneto mais rápido. Isto permitiu a construção de computadores eletrônicos muito mais rápidos, confiáveis e robustos que os a relê. O problema para o uso na automação é que os transistores não aguentam tensões e correntes tão elevadas quanto os relês.

Em 1968, a GM lançou uma especificação de um elemento de controle que pudesse substituir relês. Esta especificação pedia por:

- Facilidade de programação;
- Facilidade na manutenção (encaixe de módulos);
- Confiabilidade maior;
- Mais compacto;
- Possibilitar o envio de dados à central de processamento;
- Economicamente competitivo;
- Entradas em tensão alternada 115 V;
- Saídas em 115 V C.A. com mais que 2 A (operação das válvulas solenóides, comando para partida de motores e outros);

- Possibilitar a ampliação com um mínimo de alteração;
- Dotado de memória;

A partir desta especificação, foi desenvolvido pela Bedford Associates o primeiro PLC, posteriormente vendido pela marca MODICON – *M*ODular *D*igital *C*ONtrol*l*er. Ele usa um microcontrolador para realizar a lógica e conta com dispositivos modulares de interface de entrada e saída. A figura 3.6 mostra a arquitetura típica de um CLP.

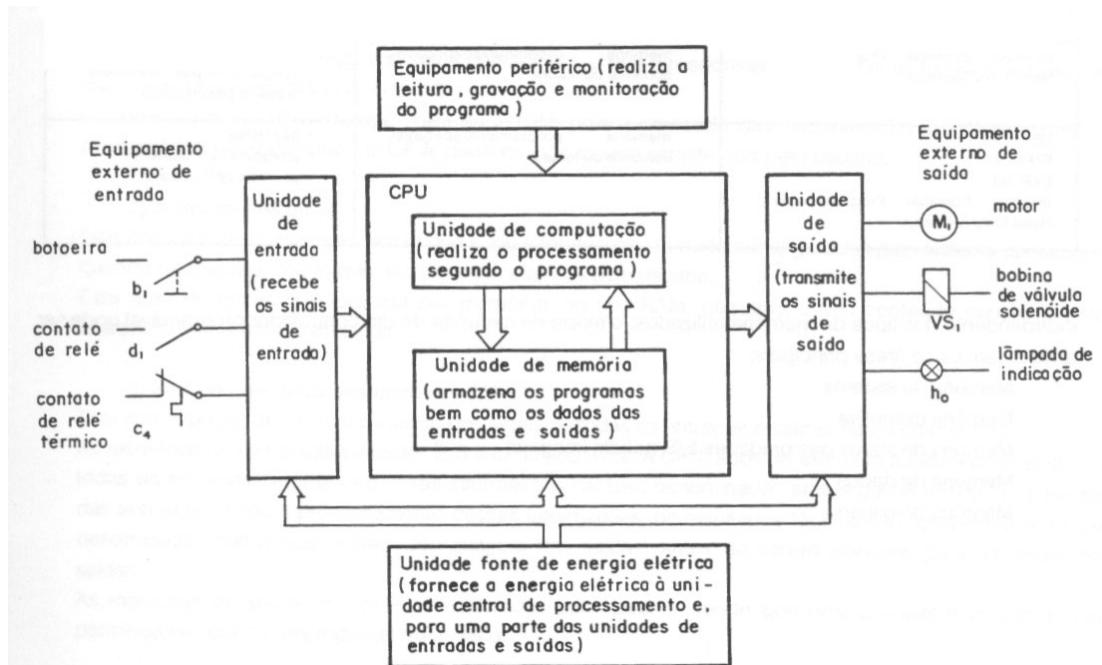


Figura 3.6: Arquitetura interna de um CLP.

As entradas e saídas dos CLPs são isoladas eletricamente, uma vez que o microcontrolador não suporta altas tensões ou correntes. Tipicamente ainda são usados relês para a implementação das saídas digitais, mas cada vez mais se tem soluções totalmente eletrônicas. A Figura 3.7 mostra circuitos típicos de entrada e saída.

3.3 IEC61131

A IEC61131, antes conhecida por IEC1131 é uma norma técnica do *International Eletrotechnical Commission*, que define um padrão para CLPs. Esta norma define vários requisitos de hardware e software para os CLPs, desde aspectos de segurança até guias de uso.

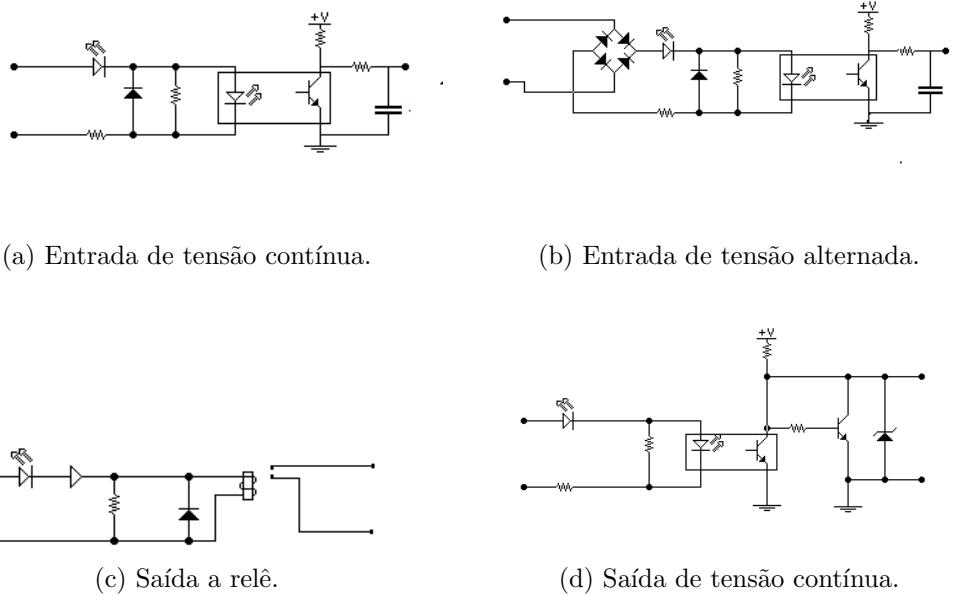


Figura 3.7: Entradas e saídas típicas de CLPs.

No CLP há um programa principal que continuamente executa a seguinte sequência:

1. Faz um auto-teste.
2. Se comunica com outros dispositivos, se houverem. Estes outros dispositivos podem ser: computador, outro CLP, IHM, etc.
3. Lê as entradas, armazenando uma cópia das entradas em posições de memória.
4. Executa o(s) código(s) do usuário.
5. Atualiza as saídas, em função das entradas e do código do usuário.

O código do usuário é composto por um conjunto de blocos em 5 possíveis linguagens, descritas na norma:

IL – Instruction List Uma linguagem de baixo nível de abstração, parecida com assembly.

ST – Structured Text Uma linguagem textual de alto nível, tal como C ou Pascal.

LD – Ladder O mesmo diagrama de circuitos a relê.

FBD – Function Block Diagram Parecido com um diagrama de portas lógicas: blocos com entradas e saídas.

SFC – Sequential Function Chart Baseado em Grafcet, uma extensão de máquina de estados.

A idéia por trás desta estrutura é que o tempo que o CLP passe executando esta sequência (conhecido por *scan time*) seja tão pequeno a ponto de poder ser desprezível. Neste contexto o código do usuário deve ser visto como tendo uma execução imediata. Obviamente isto não é verdade, já que o processador que executa este código é uma máquina de Turing – que é inherentemente sequencial. Mas o *scan time* pode ser da ordem de milissegundos ou dezenas de milissegundos, o que é bem rápido para uma grande quantidade de processos. Isto é especialmente importante quando se trabalha com a linguagem ladder, cuja estrutura remete a um circuito elétrico que é inherentemente paralelo.

Capítulo 4

Programação - arduino

O arduino é uma plataforma de microcontrolador simplificada. O nome arduino refere-se a: uma placa com um microcontrolador Atmel, uma linguagem de programação e um ambiente de desenvolvimento para esta linguagem. E também um auto-proclamado rei da Itália, mas este último não importa para nós.

A placa Arduino tem um conector USB para se ligar ao computador. Isto serve tanto para programar o microcontrolador quanto para comunicação entre os 2. Existem várias versões do arduino, pois já que é um sistema *open-source* quem quiser pode fazer sua versão diferente da placa. Vamos nos referenciar aos arduinos UNO ou outras placas compatíveis com ele.

O arduino UNO tem 4 barras de pinos fêmeas para conexão com outros dispositivos: uma com tensões de alimentação (POWER), um com 6 entradas analógicas (ANALOG IN) e 2 com um total de 14 entradas e saídas digitais (DIGITAL).

A linguagem de programação arduino é basicamente a linguagem C++ para microcontroladores ATME, mas com algumas funções e definições facilitadoras. A principal diferença entre C++ e a linguagem arduino é que não existe a função main(), mas sim as funções (ou rotinas) setup() e loop(). A função setup() é executado apenas uma vez no momento que o arduino é ligado (ou resetado) e depois o código dentro da função loop() é executado repetidamente. Com isto o esqueleto de um programa arduino fica:

Código 4.1: Esqueleto de um programa arduino.

```
void setup() {
    //codigo a ser executado no inicio
}

void loop() {
    //codigo a ser executado repetidamente
}
```

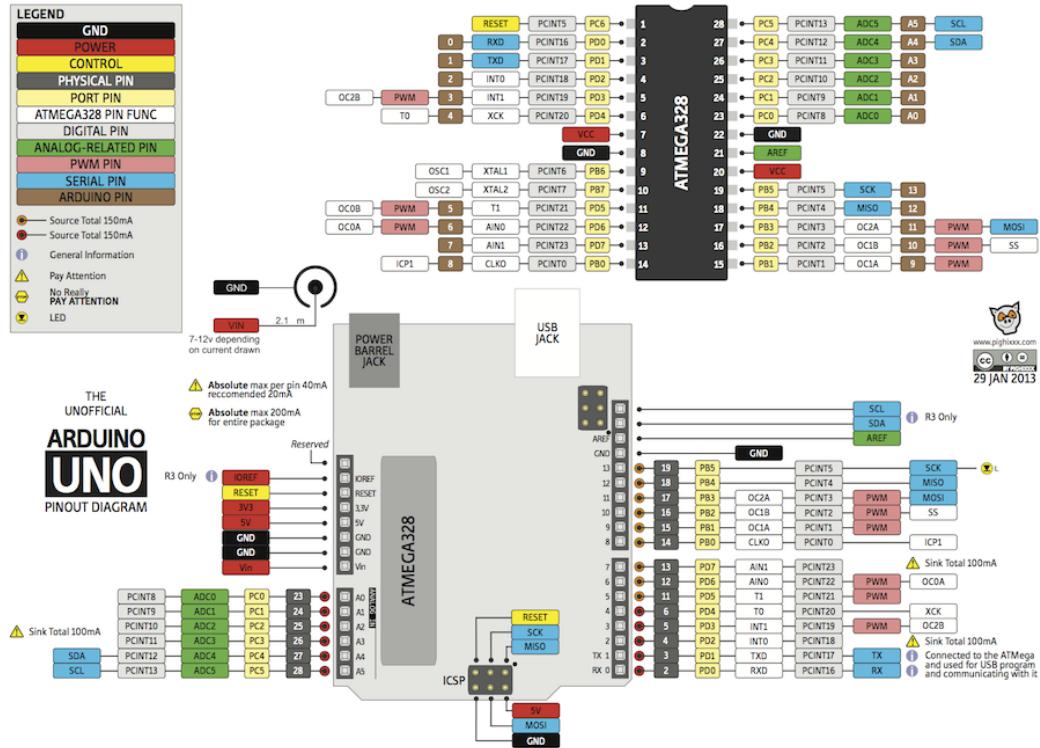


Figura 4.1: Pinagem do Arduino Uno

Lembrando que no arduino, como no C, tudo que tiver depois de // é comentário e é ignorado pelo compilador.

4.1 Piscando um led.

Vamos passar logo a um exemplo para analisar um programa arduino. As placas de arduino UNO já tem um led ligado ao pino 13, identificado por um L na placa. Podemos fazer um programa que faça este led piscar.

Código 4.2: Programa para piscar led.

```

void setup() {
    //codigo a ser executado no inicio
    pinMode(13,OUTPUT); //define o pino 13 como uma saida (led)
}

void loop() {
    //codigo a ser executado repetidamente
}

```

```

digitalWrite(13,LOW); //apaga o led
delay(500); //espera meio segundo
digitalWrite(13,HIGH); //acende o led
delay(500); //espera meio segundo
}

```

A chamada `pinMode(13, OUTPUT);` serve para definir que o pino 13 será uma saída. Obviamente isto só precisa ser feito no início do programa, logo está dentro de `setup()`. Se quiséssemos ter uma entrada digital, usariámos a mesma função, mas trocando `OUTPUT` por `INPUT`: `pinMode(pino,INPUT);`.

A função que define o valor de um pino digital é a `digitalWrite(pino,valor)`. Ela é chamada duas vezes no código 4.2 dentro de `loop()`, uma para apagar o led (gravando `LOW`) e outra para acendê-lo (gravando `HIGH`). `LOW` e `HIGH` são duas constantes, de valor 0 e 1, referentes ao zero e um lógico, respectivamente. Na prática, no sistema arduino, o `LOW` é uma tensão próxima a 0 V e o `HIGH` uma tensão próxima a 5 V.

Um detalhe é que o microcontrolador do arduino funciona numa velocidade de 8 ou 16 MHz (dependendo da versão), logo se colocássemos apenas as duas chamadas à função `digitalWrite` não veríamos o led piscar, mas teríamos a impressão que ele está aceso com metade da intensidade. Para vermos o led piscar é necessário colocar um atraso, que é justamente obtido pela função `delay(x)`, que gera um tempo morto de `x` milisegundos.

Se quiséssemos saber o valor de um pino digital que tivesse sido definido como entrada, a função seria `digitalRead(pino)`, que retornaria o valor digital naquele pino. Pode-se usar isto por exemplo, para fazer com que uma saída digital seja a cópia de uma entrada digital, como no código 4.3.

Código 4.3: Programa para acender um led em função de uma entrada digital.

```

const int pinoSaida = 13;
const int pinoEntrada = 10;

int valor;

void setup() {
    //codigo a ser executado no inicio
    pinMode(pinoSaida,OUTPUT); //define a saida (led)
    pinMode(pinoEntrada,INPUT); //define a entrada
}

void loop() {
    //codigo a ser executado repetidamente
    valor = digitalRead(pinoEntrada); //le a entrada
}

```

```

    digitalWrite(pinoSaida, valor); //e escreve na saída
}

```

No código 4.3 acrescentamos também algumas variáveis. Duas são constantes com os pinos usados. Elas facilitam a leitura do código e também facilitam caso posteriormente quisermos mudar os pinos utilizados. A outra variável armazena o valor lido da entrada, que depois é escrito na saída.

4.2 Sinais analógicos

Em contraste com os sinais digitais, os sinais analógicos são aqueles que podem assumir qualquer valor de tensão. No contexto do arduino, vamos por enquanto assumir que os sinais analógicos estão entre 0 V e 5 V.

Para valores analógicos, usamos as funções `analogWrite(pino,valor)` e `analogRead(pino)`, que, ao contrário das equivalentes digitais, são restritas a alguns pinos específicos. As entradas analógicas são identificadas pelos pinos ANALOG IN (A0 a A5 no arduino) e são ligadas a um conversor analógico/digital (A/D) do microcontrolador, que transforma estes sinais numa palavra binária de 10 bits. Como $2^{10} = 1024$, isto significa que a função `analogRead` retorna um valor entre 0 (para uma entrada de 0 V) e 1023 (para uma entrada de 5 V).

O arduino não tem um conversor D/A, logo a função `analogWrite` não gera um sinal analógico verdadeiro no pino. O que esta função faz é gerar um sinal modulado por largura de pulso - PWM (*Pulse Width Modulation*).

O sinal PWM é um trem de pulsos digital, com frequência da ordem de 500 Hz (no caso do arduino) cuja razão entre o tempo em alto e o período (conhecida como *duty cycle*) pode ser alterada pelo parâmetro passado, como mostra a figura 4.2. Se um sinal PWM é enviado a um pino com um led, ele piscará 500 vezes por segundo, o que é muito rápido para o olho humano, de modo que na prática o que se vê quando se varia o duty cycle de um sinal PWM que aciona um led é uma variação de sua intensidade. Logo um sinal PWM funciona, para muitas aplicações, como um sinal analógico.

Novamente, não são todos os pinos do arduino que conseguem gerar este sinal PWM, logo a função `analogWrite` está restrita aos pinos 3, 5, 6, 9, 10 e 11. Um outro detalhe que vale a pena ressaltar é que enquanto a função `analogRead` gera um valor entre 0 1023, a `analogWrite` recebe como parâmetro um valor entre 0 e 255 apenas.

Uma função útil do arduino para lidar com este tipo de situação é a função `map(valor, minIn, maxIn, minOut, maxOut)`, que faz uma transformação linear de valor de acordo com a seguinte equação:

$$(valor - minIn) \times \frac{maxOut - minOut}{maxIn - minIn} + minOut$$

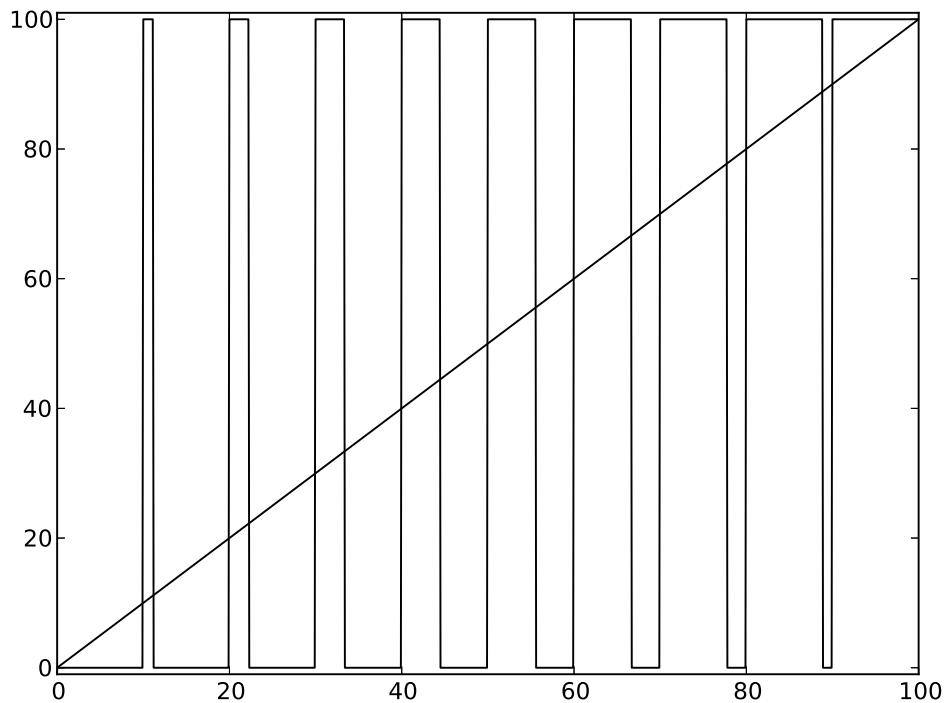


Figura 4.2: Reta modulada em largura de pulso (PWM).

A partir desta função, um código que leia o valor gerado pelo potenciômetro (em A0) e controle a intensidade do led no pino 5 poderia ser simplesmente:

```
analogWrite(5, map(analogRead(A0), 0, 1024, 0, 256));
```

Note que o valor lido pelo analogRead não precisa ser usado apenas na função analogWrite mas pode ser usado para outra finalidade, como por exemplo alterar um atraso.

4.3 Controle: for e if

Até aqui foram feitos programas puramente sequenciais, porém em vários momentos é interessante realizar operações repetidas vezes ou realizar algumas tarefas apenas em situações específicas. Para estes casos existem comandos como o **for** e o **if**. O comando for serve para tarefas repetidas. Por exemplo, se quiséssemos inicializar os pinos de 2 a 10 como saídas com valor LOW, poderíamos usar o seguinte código:

```
for (int i = 2; i < 11; i++) {
    pinMode(i, OUTPUT);
    digitalWrite(i, LOW);
}
```

O que este código faz é definir uma variável local *i* com valor inicial 2 depois ele checa se *i* é menor que 11, se for ele executa os comandos que estão entre as chaves “{” e “}”, incrementa a variável *i* (*i++*) e checa novamente. Quando *i* < 11 for falso, ele sai do laço.

O comando **if** executa um determinado código apenas se determinada condição for verdadeira. Por exemplo, para acender um led apenas se um sinal analógico for maior que a metade da escala ($512 = 1024/2$) pode-se escrever:

```
if (analogRead(A0) >= 512) {
    digitalWrite(13, HIGH);
}
```

Note que este código apenas fará alguma coisa se a condição for verdadeira. Para fazer uma coisa OU outra usa-se o comando **else** após o **if**:

```
if (analogRead(A0) >= 512) {
    digitalWrite(13, HIGH);
} else {
    digitalWrite(13, LOW);
}
```

4.4 Comunicação com o computador

Como já dito, a conexão USB do arduino serve também para a comunicação do mesmo com o computador. Do lado do computador o arduino aparece como uma porta serial e o próprio programa contém um terminal serial pelo qual é possível se comunicar com o arduino. Do lado do arduino, os pinos 0 e 1 são os pinos transmissor e receptor ligados ao USB .

A programação é feita através do objeto **Serial**. Este objeto tem vários comandos, porém para nós os que interessam neste momento são:

Serial.begin(baud) Inicializa a comunicação serial na velocidade (baud rate) indicada.

Serial.read() Retorna o valor de um byte recebido ou -1 caso não tenha sido recebido nenhum byte.

Serial.print(dado) e Serial.println(dado) Se dado for um char ou uma string (texto), envia dado. Se dado for um número, envia este número como uma string. No caso de println, é acrescentada uma quebra de linha após dado.

Serial.available() Retorna o número de bytes recebidos que ainda não foram lidos.

4.5 Ladder como uma linguagem

Apesar de parecer com um circuito, um diagrama ladder é a descrição de um código, executado da esquerda para a direita e de cima para baixo. Na prática, o ladder tem uma estrutura muti parecida com o Instruction List, portanto é interessante compararmos os 2.

Para o funcionamento do ladder ou de IL, o CLP trabalha com uma estrutura chamada pilha, que é uma memória do tipo LIFO – *last in, first out*, que pode ser visualizada como uma pilha de papel, onde o último papel colocado é o primeiro retirado. A pilha tem duas funções básicas de acesso: a load (LD), que carrega um dado de alguma posição de memória para o topo da pilha, e a store (ST), que armazena a informação no topo da pilha em alguma posição de memória. Estas posições de memória podem ser memórias internas, entradas ou saídas de um CLP. Pos simplicidade, chamemos o topo da pilha de acumulador.

Na linguagem IL, as instruções lidam diretamente com o acumulador, que passa a funcionar como uma variável implícita. Por exemplo: um código IL que define uma saída como o E de duas entradas é simplesmente:

Código 4.4: Código IL de um E lógico

```
LD %I0
AND %I1
ST %Q0
```

No código 4.4, a instrução LD %I0 carrega a entrada 0 no acumulador. Logo depois a instrução AND %I1 faz o E lógico do valor na entrada 1 com o valor no acumulador (que foi copiado da entrada 0) e guarda o resultado no próprio acumulador. É como se retirasse o valor da pilha, fizesse a operação, e guardasse de volta na pilha. Agora o acumulador contém o resultado do E lógico entre %I0 e %I1. Por último, a instrução ST %Q0 passa o valor no acumulador para a saída 0.

Este código é equivalente ao ladder da figura 4.3. Neste caso, o primeiro contato carrega a entrada 0 no acumulador. Como o segundo contato está em série, ele equivale à AND do IL. A bobina então salva o valor presente no acumulador na saída 0. Note-se então a relação que há entre a bobina do ladder e o comando ST do IL e entre o contato do ladder e os comandos LD, AND e OR do IL. Não é então de se admirar que se tenha o equivalente aos contatos normalmente fechados em IL – basta acrescentar um N aos comandos IL: LDN, STN, ANDN, ORN.

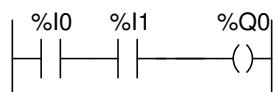


Figura 4.3: Ladder equivalente ao IL do código 4.4.

A pilha acaba sendo usada quando se tem caminhos paralelos no ladder. Analise-se por exemplo o ladder da figura 4.4. Um compilador ladder, analisando o diagrama da esquerda para a direita, vai carregar A, fazer o AND com B e, chegando à junção, checar de onde ela se origina, ele então vai carregar C, no topo da pilha acima do resultado do AND entre A e B que já está lá, fazer o AND com D, então fazer o OR entre os dois valores no topo da pilha e salvar o resultado em X. Um código IL equivalente é mostrado ao lado do ladder. Observe no IL que neste caso o comando OR não tem nenhum argumento; o que faz com que ele execute a operação com os dois últimos valores da pilha.

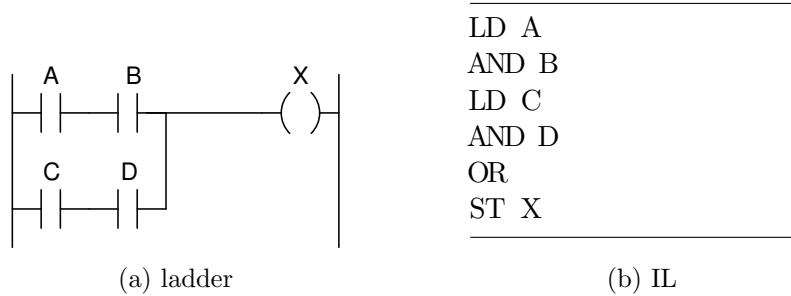


Figura 4.4: Exemplo de ladder e IL que usa a pilha para armazenar um valor intermediário.

4.6 Memória

É possível no ladder, assim como também o é usando relês, acionar uma bobina X com uma lógica que usa o contato X. Isto gera uma realimentação que tem umas propriedades interessantes. Veja por exemplo o ladder e equivalente IL da figura 4.5.



Figura 4.5: Ladder e IL implementando um latch.

Esta estrutura faz com que, se %I0 for acionado, acione-se %M0, que fica acionado daí por diante independente de %I0. %M0 só deixa de ser acionado se for acionado

$\%I1$. Isto faz com que $\%M0$ implemente uma memória de 1 bit. Esta estrutura é a base inicial das primeiras memórias de computadores eletromecânicos e é chamada de latch. Neste caso $\%I0$ funciona como o sinal de SET e $\%I1$ como o reset deste latch.

Por ser muito útil, a funcionalidade de latch acaba sendo criado diretamente, pelos comandos de set (S em IL e $\neg(S)$ em ladder) e reset, (R em IL e $\neg(R)$ em ladder). Estes comandos fazem com que o exemplo da figura 4.5 possa ser implementado mais facilmente, como mostra a figura 4.6.

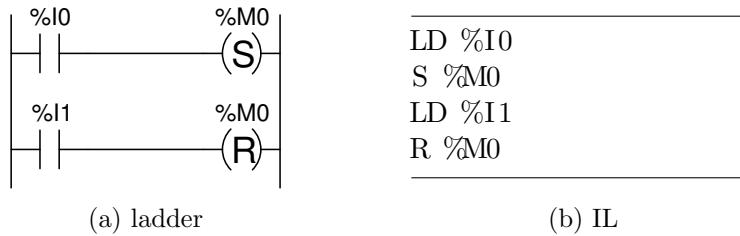


Figura 4.6: Ladder e IL implementando um latch com os comandos de set e reset.

4.6.1 Subida e descida de uma entrada.

Considere que se queira inverter uma saída cada vez que se aperte um botão. A princípio bastaria fazer o ladder da figura 4.7, que ao apertar o botão, seta a saída se ela estiver 0 ou reseta se estiver 1. Porém esta solução não funciona, pois a cada ciclo a situação da saída estará trocada e ela vai acabar oscilando enquanto a chave estiver pressionada.

É necessário então um modo de garantir a inversão da saída apenas da primeira vez que se detecta o botão apertado. Isto pode ser conseguindo armazenando o botão numa variável auxiliar *apenas ao final* do ciclo. Deste modo a variável auxiliar vai armazenar a entrada *antiga*, do ciclo anterior. Com isto podemos detectar a subida da entrada se a entrada estiver acionada mas sua cópia antiga não estiver. Isto é feito na figura 4.8.

A norma define um comando específico para ladder nesta situação, que é acionado apenas no primeiro ciclo que a variável é acionada. Este comando é identificado por um contato com um P no meio, muito embora o classicladder use um chapeuzinho, identificando a subida do sinal. Também há um comando específico para determinar a descida de um sinal, indicado ou por um N no meio do contato ou por um chapeuzinho descendo, no caso do classicladder.

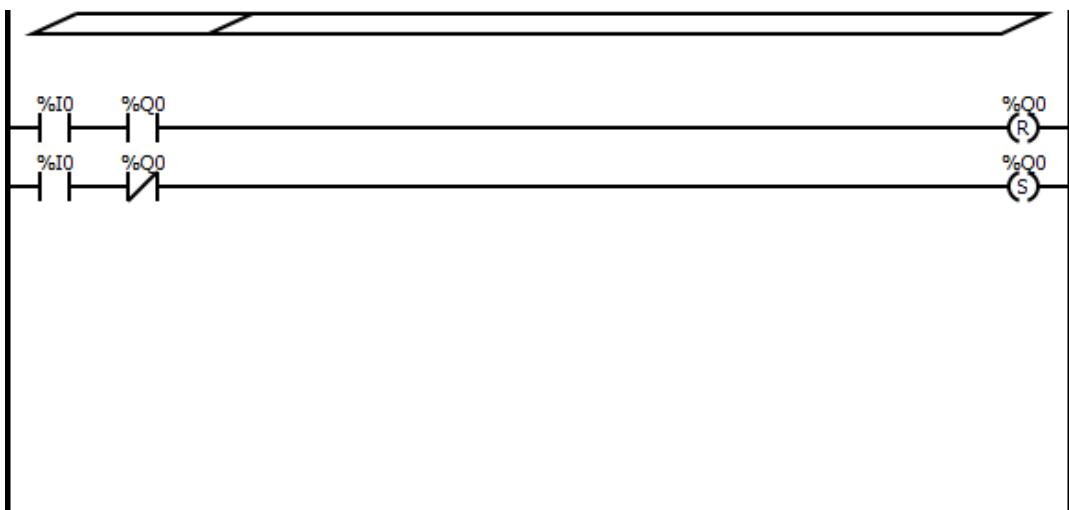


Figura 4.7: Troca de saída ao apertar um botão – solução errada.

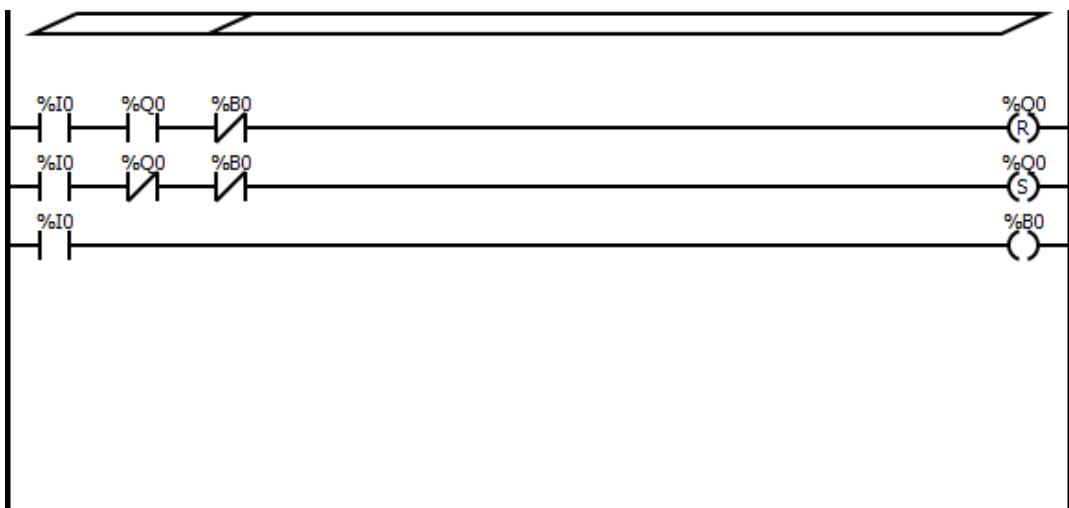


Figura 4.8: Troca de saída ao apertar um botão – solução correta.

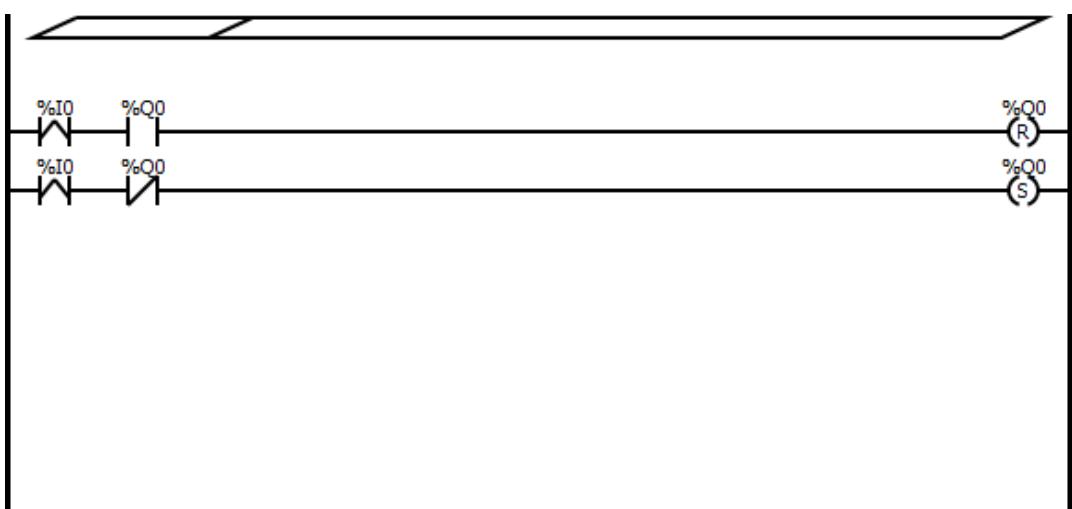


Figura 4.9: Troca de saída ao apertar um botão usando o detetor de subida.

4.7 Contadores

Contadores crescentes e decrescentes. Tem o nome %Cxx e conta com dois valores numéricos: %Cxx.V, o valor de contagem e %Cxx.P, o valor de preset. Na implementação do classicladder, o contador pode contar de 0 a 9999. O contador tem as seguintes entradas que modificam o valor de contagem V:

R – reset leva o contador para 0 quando acionado.

P – preset leva o contador para o valor de preset quando acionado.

U – up incrementa o contador na subida.

D – down decrementa o contador na subida.

O contador tem as seguintes saídas:

D – done acionado quando a contagem alcança o valor de preset (%Cxx.V = %Cxx.P).

E – empty acionado quando o contador está em zero e é decrementado (V vai para 9999). Também chamado de underflow.

F – overflow acionado quando o contador está em 9999 e é incrementado (V vai para 0).

As saídas também são disponíveis como variáveis, no formato %Cxx.X, onde xx é o número do contador e X a variável (V, P, D, E ou F). O limite de contagem pode ser resolvido com o uso destas variáveis. Por exemplo, pode-se fazer um contador que conta até 34357 da forma mostrada na figura 4.10.

4.7.1 Acesso às variáveis

As variáveis do contador, assim como outras variáveis numéricas, podem ser lidas ou definidas usando os blocos COMPARE ou OPERATE.

O bloco COMPARE retorna um valor booleano (acionado ou não acionado) definido por uma expressão matemática. Esta expressão pode conter:

- variáveis numéricas: %W0, %W3 ou símbolos definidos;
- operadores matemáticos padrões, +, -, *, /, ^ (potência), % (módulo);
- comparações, =, <, >, <=, >=, <> (diferente);
- operadores lógicos, & (and), | (or);

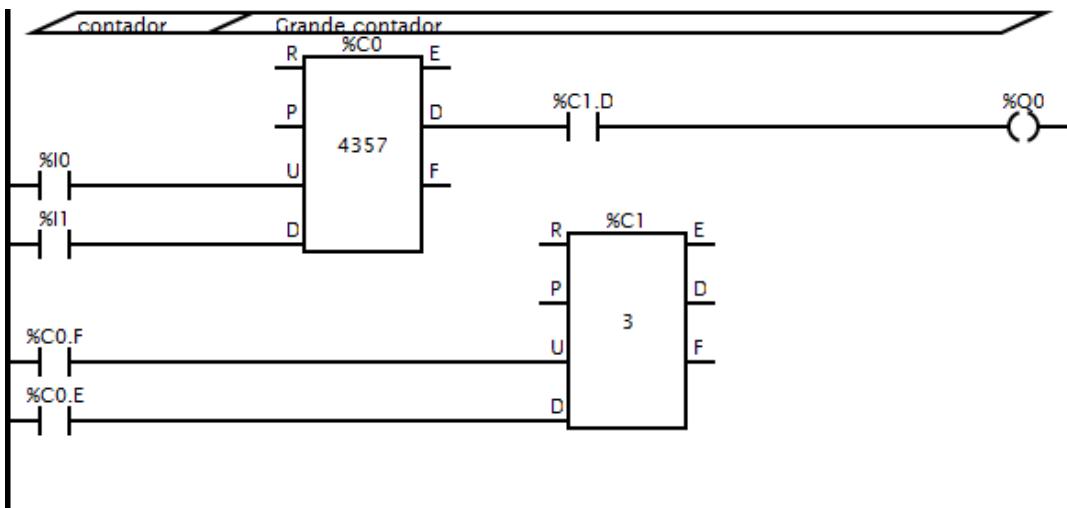


Figura 4.10: Contador com preset de 34357 feito com dois contadores.

- Funções: ABS (valor absoluta de um valor), MOY (média, em francês)/AVG (média de valores, separados por vírgula).

O bloco OPERATE é posicionado como uma bobina e recebe uma equação da forma: <variável>=valor. Se acionado acionada, como uma bobina, faz com que a variável receba o valor descrito.

Pode-se, por exemplo, definir que, quando o contador 2 contar 5, o seu valor de preset mude para 12, como mostra a figura 4.11.

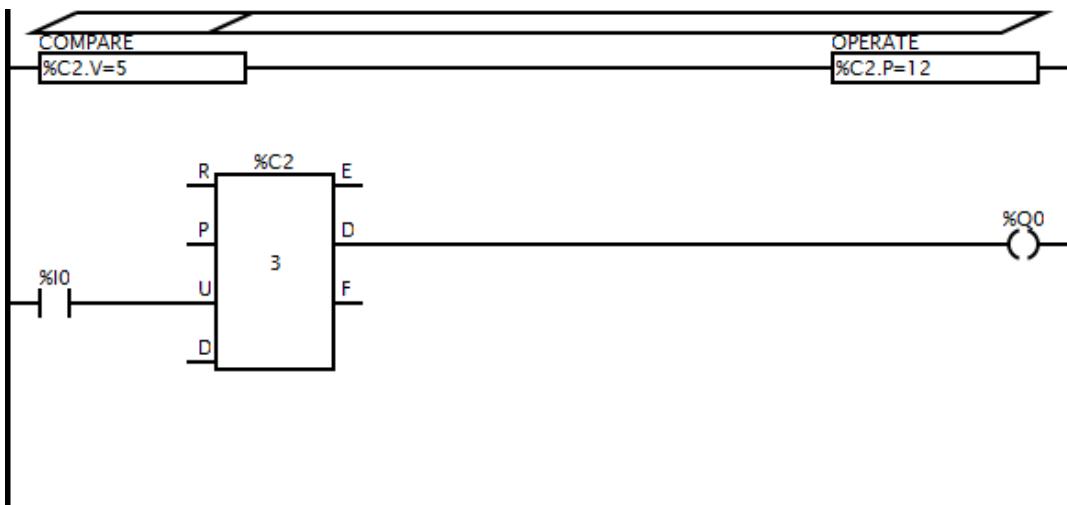


Figura 4.11: Contador tendo seu valor de preset mudado por um bloco OPERATE.

4.8 Temporizadores (timers)

O timer definido no padrão IEC61131, seguido pelo classicladder, pode ser de 3 tipos: TON, TOF ou TP. A figura 4.12 mostra um exemplo de cada um destes tipos.

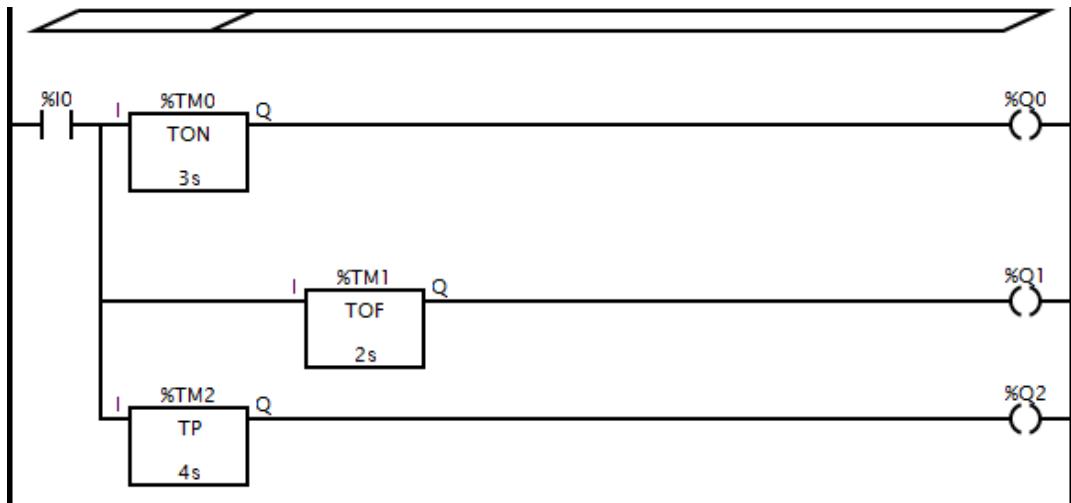


Figura 4.12: Timers TON, TOFF e TP.

O temporizador tipo TON causa um atraso no tempo de subida do sinal. O TOF causa um atraso no tempo de descida do sinal. O TP causa um pulso de tempo determinado iniciado pela subida do sinal de entrada. Isto pode ser visto na figura 4.13.

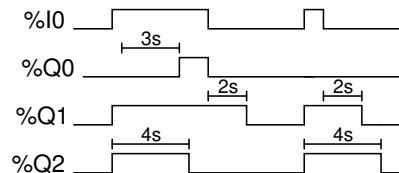


Figura 4.13: Diagrama temporal com os sinais em relação ao ladder da figura 4.12.

4.9 plcLib

A biblioteca plcLib (<http://www.electronics-micros.com/software-hardware/plclib-arduino/>) define um conjunto de funções muito parecidas com as da linguagem *Instruction List*, e que podem ser facilmente transcritas para a linguagem Ladder. Na configuração padrão, chamada pela função setupPLC() este sistema define 4 entradas (X0 a X3, usando A0 a A3) e quatro saídas (Y0 a Y3, usando 3, 5, 6 e 9).

Vamos usar um shield de arduino que já tem 3 botões de entrada, 4 leds, uma busina e um potenciômetro: o Multi-function shield (MFS). Usando o MFS, as entradas correspondem ao potenciômetro (A0) e aos 3 botões, logo estão bem adequadas. As saídas correspondem ao buzzer (3) e a três conectores de servo, porém não temos nenhuma saída ligada aos leds. Para definir todas entradas e saídas de uma vez, fazemos:

Código 4.5: Definição de função e constantes para usar o o Multi-function Shield com o plcLib.

```
// Definicao das constantes dos pinos
const int VR = A0;
const int S1 = A1;
const int S2 = A2;
const int S3 = A3;
const int D1 = 13;
const int D2 = 12;
const int D3 = 11;
const int D4 = 10;
const int LS1 = 3;
const int Q1 = 5;
const int Q2 = 6;
const int Q3 = 9;
const int Q4 = A5;

void setupShield(){ // Funcao para inicializar
    pinMode(VR,INPUT);
    pinMode(S1,INPUT);
    pinMode(S2,INPUT);
    pinMode(S3,INPUT);
    pinMode(D1, OUTPUT);
    pinMode(D2, OUTPUT);
    pinMode(D3, OUTPUT);
    pinMode(D4, OUTPUT);
    pinMode(Q1, OUTPUT);
    pinMode(Q2, OUTPUT);
```

```

pinMode(Q3, OUTPUT);
pinMode(Q4, OUTPUT);
pinMode(LS1, OUTPUT);
// Valores iniciais para apagar os leds e nao soar a buzina
digitalWrite(D1,HIGH);
digitalWrite(D2,HIGH);
digitalWrite(D3,HIGH);
digitalWrite(D4,HIGH);
digitalWrite(LS1,HIGH);
}

```

Deste modo basta chamarmos setupShield() no setup que configuramos nossa entradas e saídas. Infelizmente neste shield, os botões geram 0 quando apertados e 1 quando ligados e tanto os leds quanto a buzina acionam com nível lógico baixo, o que faz com que nossas entradas e saídas tenham que ser invertidas. Isto porém não impede o funcionamento do circuito.

4.9.1 Acumulador

Os CLPs trabalham com o conceito de acumulador (que na implementação do plcLib tem o nome scanValue), que é uma variável implícita. As funções pegam implicitamente o valor a ser trabalhado do acumulador e salvam o resultado de volta nele. O plcLib define as funções in(entrada), inNot(entrada), out(saida) e outNot(saida), equivalentes às LD, LDN, ST e STN, de Instruction List, respectivamente. As duas primeiras pegam o valor de uma entrada e guardam no acumulador, enquanto que as 2 últimas pegam o valor do acumulador e colocam na saída. O Not (N) no final indica que estes valores são invertidos. Vejamos como exemplo o código 4.6:

Código 4.6: Código simples de leitura de cópia de entradas para saídas.

```

void loop() {
    in(S1);      // Le chave 1
    out(D1);     // manda para led 1

    inNot(S2);   // Le chave 2 (invertida)
    out(D2);     // manda para led 2

    inNot(S3);   // Le chave 3 (invertida)
    outNot(D3);  // manda para led 2 (invertido)
}

```

Neste caso, D1 fica sendo uma cópia de S1, D2 o valor invertido de S2. Dá para imaginar o que acontece com D3.

É bom lembrar que no multi-function shield as chaves são conectadas ao terra, enquanto que os leds são conectados à 5V. Isto faz com que quando apertada, a chave gera um 0 lógico (não um 1) e que o led acende quando se envia um 0 lógico (e não um 1). Por isto é interessante para trabalharmos com este shield fazermos `inNot(Sn)`, que resulta em 1 se a chave estiver apertada, e `outNot(Ln)`, que acende o led quando o valor no acumulador for 1.

4.9.2 Funções lógicas

Mas apenas carregar um único valor não é tão interessante. Bem mais útil é montar uma relação lógica entre valores. Para isto servem as funções `andBit`, `orBit`, `xorBit`, `andNotBit` e `orNotBit`. Cada uma destas funções faz uma operação lógica entre o acumulador e a variável indicada, salvando o resultado no acumulador. A tabela 4.1 mostra as tabelas verdadeas de cada uma delas.

Tabela 4.1: Tabela verdade das funções lógicas definidas em `plcLib`.

scanValue	ent	scanValue (após operação)				
		andBit	orBit	xorBit	andNotBit	orNotBit
0	0	0	0	0	0	1
0	1	0	1	1	0	0
1	0	0	1	1	1	1
1	1	1	1	0	0	1

Colocando estas funções em cascata é possível criar lógicas bem interessantes, como por exemplo, soar o alarme se S1 ou S2 forem pressionados ao mesmo tempo que S3 for pressionado.

Código 4.7: Aciona a buzina em função das chaves.

```
inNot(S1); // Le se chave 1 apertada
orNotBit(S2); // Le se chave 2 apertada e faz um OU logico
andNotBit(S3); // Le se chave 3 apertada e faz um E logico
outNot(LS1); // Se condicao satisfeita , aciona a buzina
```

O problema do uso do acumulador é que às vezes uma situação exige uma lógica mais complexa do que o acumulador permite. Por exemplo: como acender D2 se a maioria dos botões estiver pressionado? (Problema do voto majoritário) Existem 2 formas de resolver este problema: pilha ou variáveis;

A pilha é como a maioria dos CLPs trabalham: ao invés de ter um único acumulador, ele tem um número finito de posições de memória que ele usa e todo comando usa implicitamente o topo da pilha. No caso da implementação do `plcLib`,

a pilha pode ser definida separadamente como um objeto da classe Stack. Logo, se fizermos Stack pilha; definimos uma pilha de nome pilha.

Os principais comandos de acesso à pilha são o push(), que passam o valor do acumulador para o topo da pilha, e o pop(), que tira o valor do topo da pilha e passa para o acumulador. Outros comandos úteis para a lógica são o andBlock(), que faz o E do valor no topo da pilha com o acumulador, salvando no acumulador e o orBlock(), que faz a mesma coisa com o OU lógico. Usando a pilha é possível resolver o problema da maioria dos botões da seguinte forma:

Código 4.8: Solução do voto majoritário com uso de pilha.

```
void loop(){\lstinline|
    inNot(S1);
    andNotBit(S2);
    andBit(S3);
    pilha.push();
    inNot(S1);
    andBit(S2);
    andNotBit(S3);
    pilha.push();
    in(S1);
    andNotBit(S2);
    andNotBit(S3);
    pilha.push();
    inNot(S1);
    andNotBit(S2);
    andNotBit(S3);
    pilha.orBlock();
    pilha.orBlock();
    pilha.orBlock();
    outNot(D2);
}
```

Outra possibilidade é simplesmente definir variáveis auxiliares para receberem os valores intermediários. Neste caso as variáveis precisam ser do tipo **unsigned int**.

Código 4.9: Solução do voto majoritário com uso de variável.

```
int temp1;
void loop(){
    inNot(S1);
    andNotBit(S2);
    andBit(S3);
    out(temp1);
    inNot(S1);
```

```

    andBit(S2);
    andNotBit(S3);
    orBit(temp1);
    out(temp1);
    in(S1);
    andNotBit(S2);
    andNotBit(S3);
    orBit(temp1);
    out(temp1);
    inNot(S1);
    andNotBit(S2);
    andNotBit(S3);
    pilha.orBlock();
    pilha.orBlock();
    pilha.orBlock();
    orBit(temp1);
    outNot(D1);
}

```

Note-se que não se usou a melhor estratégia lógica para ambas soluções apresentadas. Fica como exercício resolver este problema de forma mais compacta.

4.9.3 Memória

Até o momento usamos apenas a chamada lógica combinacional, onde a saída depende apenas da entrada. Porém é bem comum a situação de querermos que a saída fique num determinado estado em função da sua história pregressa.

Um exemplo simples: Como acionar LS1 apertando S1 e pará-lo apertando S2? O estado de LS1 depende então de qual foi o último botão apertado.

É possível implementar este tipo de memória a partir de lógica combinacional usando realimentação. Ou seja, devemos ler a saída do sistema como sendo entrada, o que, apesar de estranho, é perfeitamente possível. O código 4.10 mostra justamente esta implementação.

Código 4.10: Implementação de latch com lógica combinacional.

```

inNot(S1);      // Se chave S1 apertada
orNotBit(LS1);  // ou se LS1 ja esta acionado
andBit(S2);     // mas apenas se S2 nao estiver apertada
outNot(LS1);    // aciona LS1

```

Como esta função é bastante utilizada, ela recebeu o nome de *latch* (tranca, ferrolho) e tem funções específicas, para deixar uma variável em 1 (função `set()`) ou 0 (`reset`) dali em diante, como mostrado no código 4.11.

Código 4.11: Funções para uso de latch.

```
inNot(S1);      // Se chave S1 apertada
reset(LS1);     // Seta LS1 (aciona dai em diante)
andBit(S2);     // Se S2 estiver apertada
set(LS1);       // Reseta LS1 (desliga dai em diante)
```

4.9.4 Temporizadores

O arduino já tem uma função padrão `delay`, que causa a paralisação da execução por um determinado tempo. O problema de `delay` é que ele para o programa todo, o que pode ocasionar problemas até de segurança.

Os temporizadores definidos na `plcLib` permitem gerarmos atrasos em sinais específicos, sem mexer nos demais. O `timerOn` atrasa a subida de um sinal, enquanto que o `timerOff` atrasa a descida de um sinal, como pode ser visto no exemplo abaixo. Outro tipo de temporizador é o `timerPulse`, que na subida do sinal de entrada gera um pulso por um tempo determinado. `timerPulse` e `timerOff` são bem parecidos, com a diferença que o último começa a medir o tempo a partir da descida da entrada, enquanto que o primeiro mede a partir da subida da entrada. Cada função destas recebe como parâmetro a variável (do tipo `unsigned long`) que contará o tempo e o tempo final a ser contado. Logo para cada temporizador é necessário ter uma variável para armazenar o tempo utilizado.

Código 4.12: Exemplos de uso de temporizadores.

```
unsigned long TIMER0 = 0;
unsigned long TIMER1 = 0;
unsigned long TIMEa = 0;
void loop(){
    inNot(S1);      // Se chave S1 apertada
    timerOn(TIMER0,2000); // atrasa subida por 2 segundos
    outNot(D1);
    inNot(S2);      // Se chave S2 apertada
    timerOff(TIMER1,2000); // atrasa descida por 2 segundos
    outNot(D2);
    inNot(S3);      // Se chave S3 apertada
    timerPulse(TIMERa, 2000); // gera pulso de 2 segundos
    outNot(D3);
}
```

Um outro temporizador interessante é o `timerCycle`, que enquanto o acumulador for 1, gera um sinal alternado definido por 2 tempos: o tempo em alto e o tempo em baixo. Muito útil para alarmes.

Código 4.13: Exemplos de uso timerCycle.

```
unsigned long TempoLOW = 0;
unsigned long TempoHIGH = 0;
void loop(){
    inNot(S1);
    timerCycle(TempoLOW,1300,TempoHIGH,100);
    outNot(LS1);
}
```

4.9.5 Contadores

Um contador incrementa ou decremente uma variável de acordo com o número de eventos que recebe. Na implementação do plcLib, estes eventos são subidas (ir de 0 para 1) em suas entradas.

Para os contadores, cria-se um objeto do tipo Counter, que tem um valor interno de preset presetValue, uma variável de contagem count, 4 entradas countUp, countDown, preset, clear e 2 saídas upperQ e lowerQ. As regras do contador são:

- Uma subida em countUp incrementa o valor de count, se for menor que preset.
- Uma subida em countDown decrementa o valor de count, se for maior que 0.
- Uma subida em clear faz count igual a 0.
- Uma subida em preset faz count igual a presetValue.
- Se count for igual a 0, lowerQ retorna 1.
- se count for igual a preset, upperQ retorna 1.

Código 4.14: Exemplo de uso do contador.

```
Counter ctr(10);           // Contador na faixa 0-10, começando em zero
unsigned long TIMER0 = 0;   //
unsigned long TIMER1 = 0;   //

void loop() {
    inNot(S1);             // Se S1 apertada
    timerOn(TIMER0, 10);    // 10 ms de atraso para debounce
    ctr.countUp();          // incrementa ctr.
    inNot(S2);              // Se S2 apertada
    timerOn(TIMER1, 10);    // 10 ms de atraso para debounce
```

```

ctr.countDown();           // decrementa ctr.
inNot(S3);                // Se S3 apertada
ctr.clear();               // zera o contador.
ctr.lowerQ();              // Se zero ,
outNot(D1);                // Se 10,
ctr.upperQ();             // Se 10,
outNot(D2);
}

```

É possível ainda criar um contador que inicializa no valor de preset. Tomando o exemplo do código 4.14, ao invés de fazer Counter ctr(10);, faria-se Counter ctr(10,1);

Capítulo 5

Linguagem grafset

GRAphe Fonctionnel de Commande Etape/Transition - Grafo funcional de comando etapa/transição, é um formalismo matemático tal como uma máquina de estados. Tem origem na chamada máquina de estados, mas com mais funcionalidades, tais como a possibilidade de ter estados em paralelo e de ter macro-estados.

Um diagrama grafset é composto dos seguintes elementos: estados, transições, ações, divergências e convergências E e divergências e convergências OU.

O básico do grafset é a sequência estado-transição-estado, como mostra na figura 5.1. Ela apresenta um diagrama grafset com 3 estados (os quadrados) e três transições (os retângulos verde e vermelhos). O estado 0 é o estado inicial, o que é indicado pela linha dupla, porém neste momento o estado ativo é o Estado 1, indicado pela ficha. As transições tem condições que, quando verdadeiras, as ativam. Neste caso uma transição ativa é representada pela cor verde, enquanto uma inativa é representada pela cor vermelha.

A evolução do sistema descrito em grafset é dada pelo disparo das transições. Um transição dispara quando:

- ela está ativa e
- todos os estados anteriores a ela estão ativos.

Quando uma transição dispara ela desativa todos os estados anteriores a ela (os estados acima) e ativa todos os posteriores (os abaixo). Por exemplo, se na figura 5.1 a condição A se tornasse verdadeira, como o único estado anterior, o Estado 1 está ativo, esta transição dispararia, desativando Estado 1 e ativando Estado 2. Esta sequência é mostrada na figura 5.2.

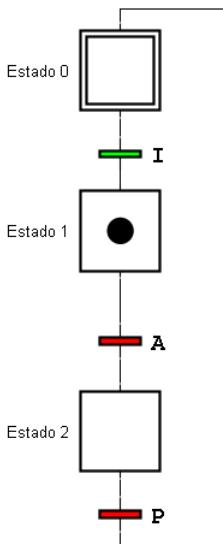


Figura 5.1: Exemplo de um diagrama grafcet simples.

5.1 Divergências e convergências

No grafcet, pode-se ter um estado com mais de uma transição de saída, que pode levar para estados diferentes de acordo com qual transição dispare primeiro. A isto se chama uma *divergência OU*, pois separa em dois caminhos excludentes. Da mesma forma, duas (ou mais) transições podem levar para o mesmo estado, o que se chama de convergência OU. A figura 5.3 mostra um exemplo em que o sistema pode evoluir por dois caminhos diferentes, com o uso de divergência e convergência OU.

Note-se na figura 5.3 que o Estado 1 é obrigatoriamente transitório, pois ou A é verdadeiro e o sistema evolui imediatamente para o Estado 2 ou A é falso e o sistema ativa o Estado 3. É possível definir transições que estejam ativas ao mesmo tempo, neste caso o sistema evolui para a transição com maior prioridade – ou a mais à esquerda ou se tiver uma prioridade explícita.

Diferente de uma máquina de estados, no grafcet é permitido que uma transição acione mais que um estado. Neste caso o sistema se divide em caminhos paralelos. Isto é chamado de divergência E e indicado por duas linhas horizontais, para diferenciar da divergência OU. Da mesma forma, uma convergência E liga vários estágios a uma única transição, transformando caminhos paralelos num único caminho. A figura 5.4 mostra um exemplo de divergência e convergência E.

Note que, neste caso, mesmo com P ativo, o sistema não desativa o Estado 2a nem ativa Estado 0, pois como o Estado 2b não está ativo, a transição P não pode disparar.

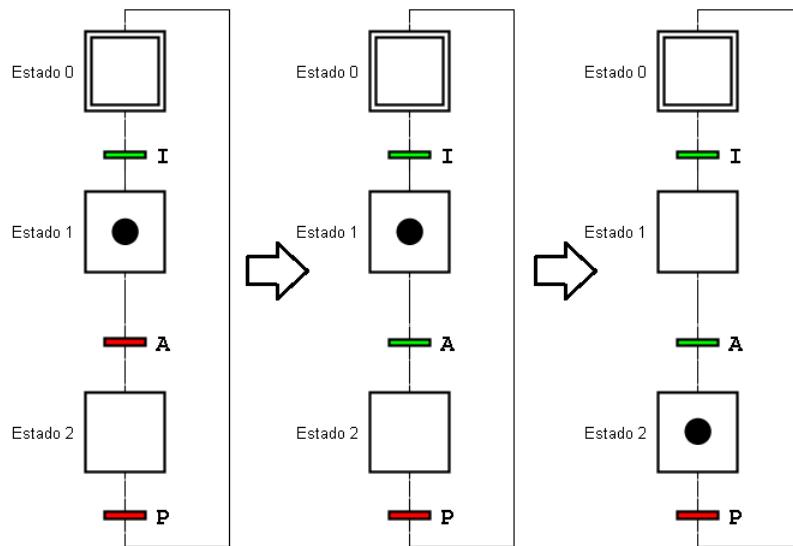


Figura 5.2: Sequência de disparo da transição A

5.2 Ações

Outra característica do grafcet é que podem ser definidas ações, ligadas a estados. O que exatamente compõem uma ação depende da implementação do grafcet. Peguemos dois exemplos: o JGrafchart e a SFC, que é a descrição de grafcet definida pela norma IEC61131.

No JGrafchart, uma ação pode ser uma definição de valor (um *assignment*), da forma variavel = expressao, uma chamada de subrotina (*call*) ou uma variável booleana (*bool*), que fica verdadeira enquanto o estado estiver ativo. Um qualificador de uma letra define o momento em que a ação é executada:

S assignment|call; Executa apenas uma vez, quando a etapa é ativada.

X assignment|call; Executa apenas uma vez, quando a etapa é desativada.

P assignment|call; Executa a cada ciclo enquanto a ação estiver acionada.

N bool; Aciona a variável booleana enquanto a etapa estiver acionada.

A assignment|call; Executa apenas uma vez, se a etapa for desativada por uma transição de exceção (apenas definida no JGrafchart).

Note-se que a ação **N bool;** é equivalente a **P bool=1;** e é presente para facilitar a escrita, já que tal operação é feita muitas vezes.

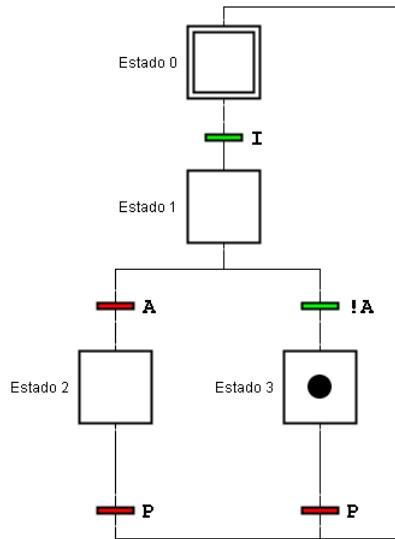


Figura 5.3: Exemplo de grafcet com divergência e convergência OU. O $!A$ indica que a transição está ativa quando A for falso.

Toda etapa tem uma variável booleana x ($<\text{nome_da_etapa}>.x$), que aciona quando a etapa é acionada, e uma variável de tempo em décimos de segundo t ($<\text{nome_da_etapa}>.t$) e em segundos s ($<\text{nome_da_etapa}>.s$).

As ações do SFC são do formato **qualificador** **funcão|variável** **finalizado**. **finalizado** é uma variável opcional no caso de se usar uma função, que aciona quando a função termina, o que a torna útil para ser utilizada na transição de saída da etapa.

O qualificador pode ser qualquer um dos descritos na tabela 5.1. Pode-se não especificar o qualificador, onde o comportamento fica o mesmo de se usar o qualificador N. No caso da ação ser uma função, ele é executada sempre que ficar ativa. Todos os qualificadores que tem tempo acompanham um valor de tempo dado ou por uma variável ou por uma constante da forma $T\#4k$, onde k pode ser t (décimo de segundo), s (segundo), m, (minuto) ou h (hora).

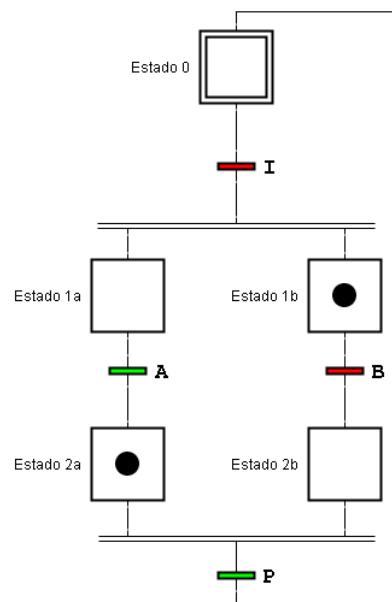


Figura 5.4: Exemplo de grafcet com divergência e convergência E.

Tabela 5.1: Qualificadores do SFC

Qualificador	Descrição
N	Ativa a etapa estiver acionada.
S	Set – ativa e mantém ativado mesmo depois de sair da etapa.
R	Reset – desativa.
P	Ativa apenas no primeiro ciclo que a etapa estiver ativa.
L tempo	Ativa pelo tempo determinado enquanto a etapa estiver acionada.
SL tempo	Ativa pelo tempo determinado independente da etapa ainda estar acionada.
D tempo	Ativa após o tempo determinado enquanto a etapa estiver acionada.
SD tempo	Seta após o tempo determinado independente da etapa ainda estar acionada.
DS tempo	Seta após o tempo determinado se a etapa ainda estiver acionada.

5.3 Níveis

O Grafcet pode ser definido em 2 níveis de abstração: o 1 e o 2. O nível 1 serve como uma ferramenta de desenvolvimento, para analisar a partir do problema como um todo a sequência de etapas, as ações a serem realizadas em cada etapa e as condições de transição de uma etapa para outra. No nível 1 as ações e transições são descritas de forma ampla, para permitirem uma melhor visualização do processo pelo desenvolvedor. O resultado final é uma descrição dos requisitos daquelas etapas e transições e não serve como uma linguagem para programar um sistema.

Já o Grafcet nível 2 é propriamente uma linguagem de programação, com diferentes implementações. Uma delas é o SFC - *Sequential Function Chart*, descrita no padrão IEC1131-3 para programação de CLPs. No SFC, cada ação corresponde a uma atuação nas saídas ou variáveis internas do CLP e cada transição corresponde a um valor binário obtido no CLP seja de entradas, seja de comparações de valores analógicos ou de tempo.

Desta forma o principal uso do grafcet é num projeto top-down, onde a partir do problema se desenvolve a sequência a ser seguida no grafcet nível 1, a partir deste se definem todas as entradas, saídas e condições necessárias para o controle automático daquela sequência e então programa-se o sistema usando grafcet nível 2.

Tomemos como exemplo uma tarefa relativamente simples: cozinhar um ovo. Considere-se o sistema da figura 5.5, consistindo de uma panela, uma entrada de água, uma boca de fogão a gás e um mecanismo que solte o ovo na água.

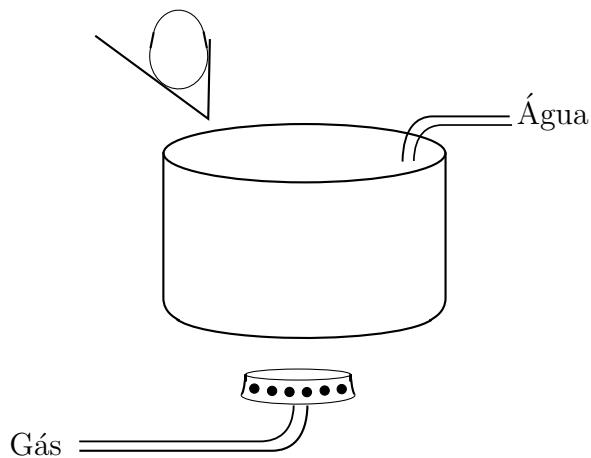


Figura 5.5: Sistema para cozinhar ovo sem sensores ou atuadores.

Podemos inicialmente fazer um grafcet considerando que tudo dá sempre certo: ao iniciarmos o sistema, a panela enche de água, o ovo é colocado, o fogo acendido

e aguarda-se até o ovo ficar pronto, de onde se retira a panela para tirar o ovo. Esta sequência é descrita pelo grafctet da figura 5.6.

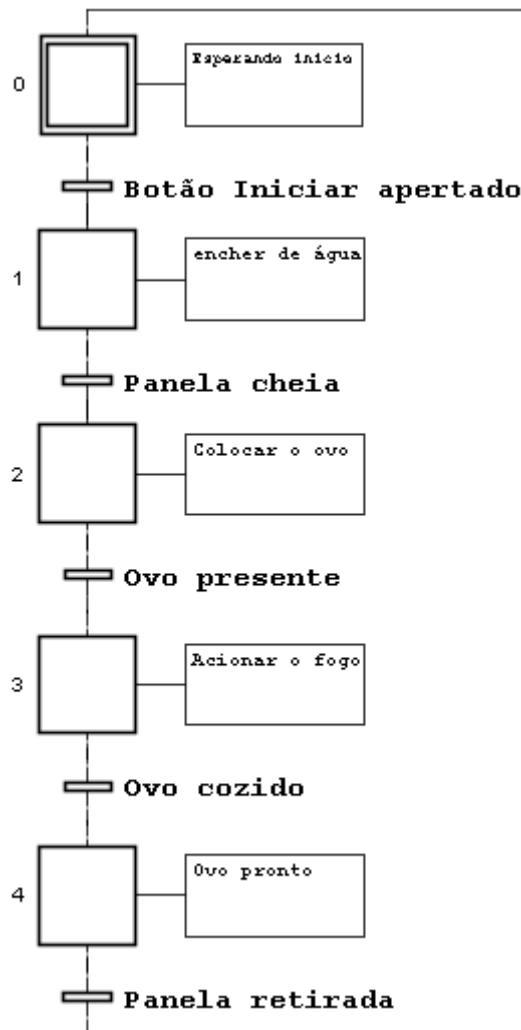


Figura 5.6: Diagrama grafcet nível 1 para cozinar um ovo de forma otimista.

Porém, problemas acontecem. Pode-se, por exemplo, esquecer de colocar a panela de volta antes de iniciar, faltar fogo ou acabar o ovo. Colocando todas estas condições a mais e definindo o que fazer se elas acontecerem o grafcet cresce para o apresentado na figura 5.7.

Com base neste diagrama, pode-se analisar agora quais são os sensores e atuadores necessários para esta tarefa, mostrados na figura 5.8. São eles:

S_PP Sensor de presença para detectar a panela.

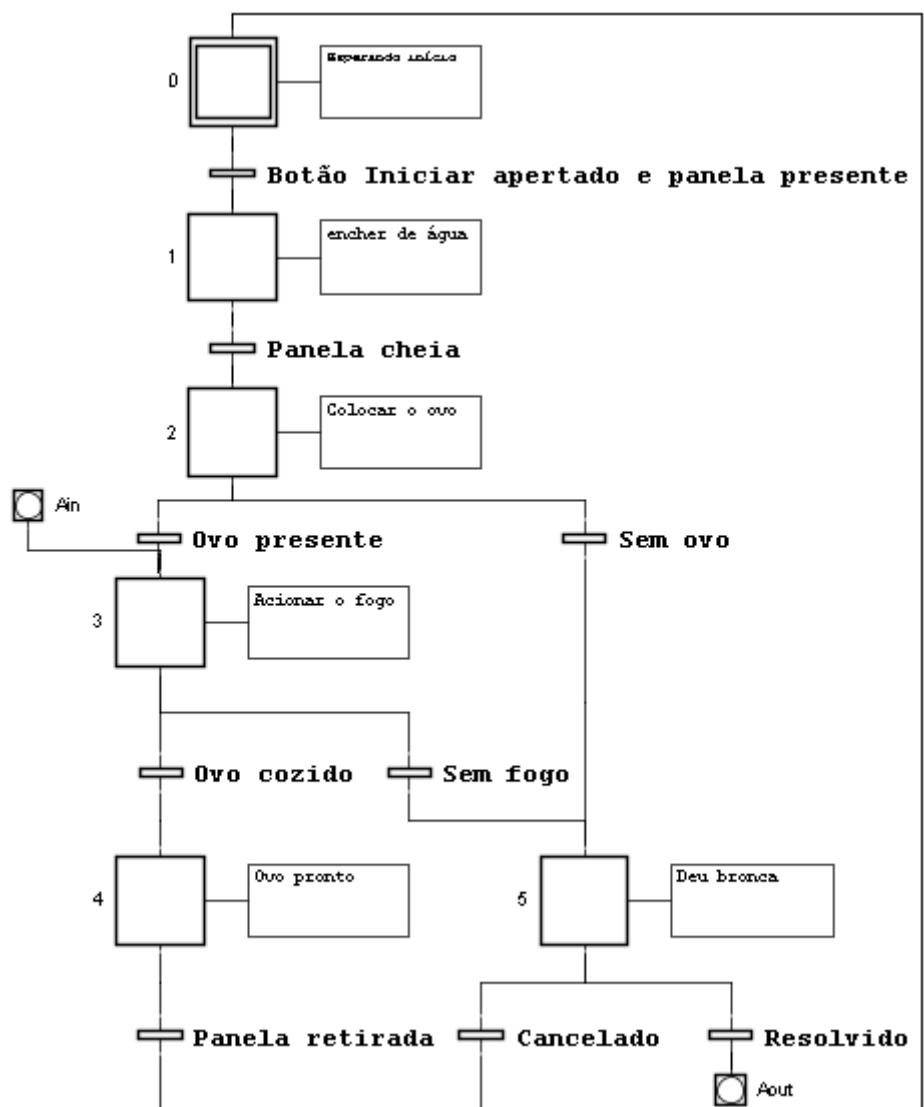


Figura 5.7: Diagrama grafcet nível 1 para cozinhar um ovo.

S_N Sensor de distância para obter o nível de água.

S_G Sensor de gás.

B_I Botão de início.

B_OK Botão de OK, para continuar se der bronca.

B_C Botão de cancelamento.

PO Solenoíde para abrir a portinhola do ovo.

VA Válvula de controle da água.

VG Válvula de controle do gás.

CENTELHA para ligar o fogo.

ALARME para indicar que deu errado.

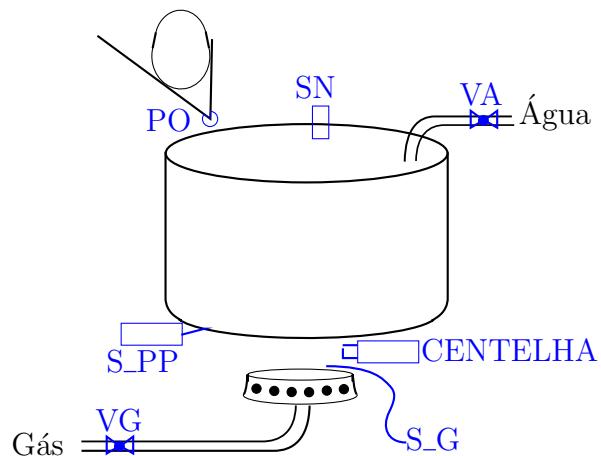


Figura 5.8: Sistema para cozinhar ovo com sensores e atuadores.

Então com base nos sensores e atuadores presentes, podemos fazer o grafctet nível 2 (figura 5.9, descrevendo as ações e transições de acordo com estas entradas e saídas.

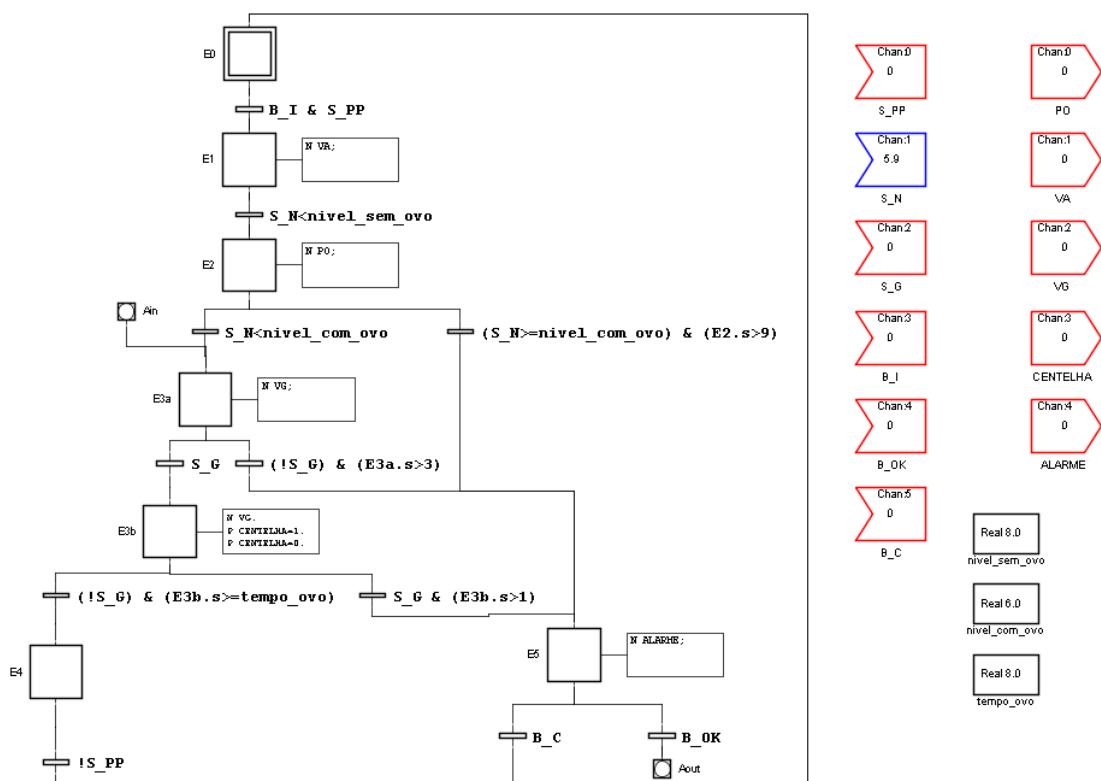


Figura 5.9: Diagrama grafcet nível 2 para cozinhar um ovo.

Capítulo 6

SCADA – Supervisory Control and Data Acquisition

6.1 Arquitetura de sistemas SCADA e interfaceamento com níveis de automação

6.2 Funcionalidades principais de sistemas SCADA

Capítulo 7

Sistema SCADA MANGO

Mango é uma solução SCADA/HMI open source que roda em um sevidor internet java. Logo toda interface dele é através de um navegador. Muito embora ele seja open source, existem versões aprimoradas dele que apresentam umas facilidades a mais, mas a versão básica já permite fazer um sistema SCADA completo.

7.1 Instalação

Vide <http://infiniteautomation.com/wiki/wiki.php>, a instalação é simplesmente descompactar um arquivo zip e rodar um script para abrir o programa. Basta ter o java 1.7 instalado.

7.2 Uso

Ao rodar o mago, ele abre a tela mostrada na figura 7.1. A página do mango pode ser acessada novamente pelo endereço <http://localhost:8080>. O login padrão é **admin**, senha **admin**.

O topo da tela do mango pode ser visto na figura 7.2. Na parte mais superior tem os indicadores de eventos e alarmes, seguido dos links para as diversas áreas do mango.

Um dos primeiros pontos a ser definido é o das fontes de dados do sistema. Chega-se nisto clicando no ícone *data sources*, vide figura 7.3. São possíveis diversos tipos diferentes de fontes de dados, mas para nós no momento os que vão interessar são o **Modbus I/P**, o **Modbus Serial** e o **Virtual Data Source**, este último principalmente para simulação.

Para cada fonte de dados, são criados pontos (*data points*), correspondentes a uma tag. No caso de uma fonte virtual, há opções quanto ao tipo de dado (binário, multi-estado, numérico, texto) e quanto ao valor automático, vide figura 7.4.

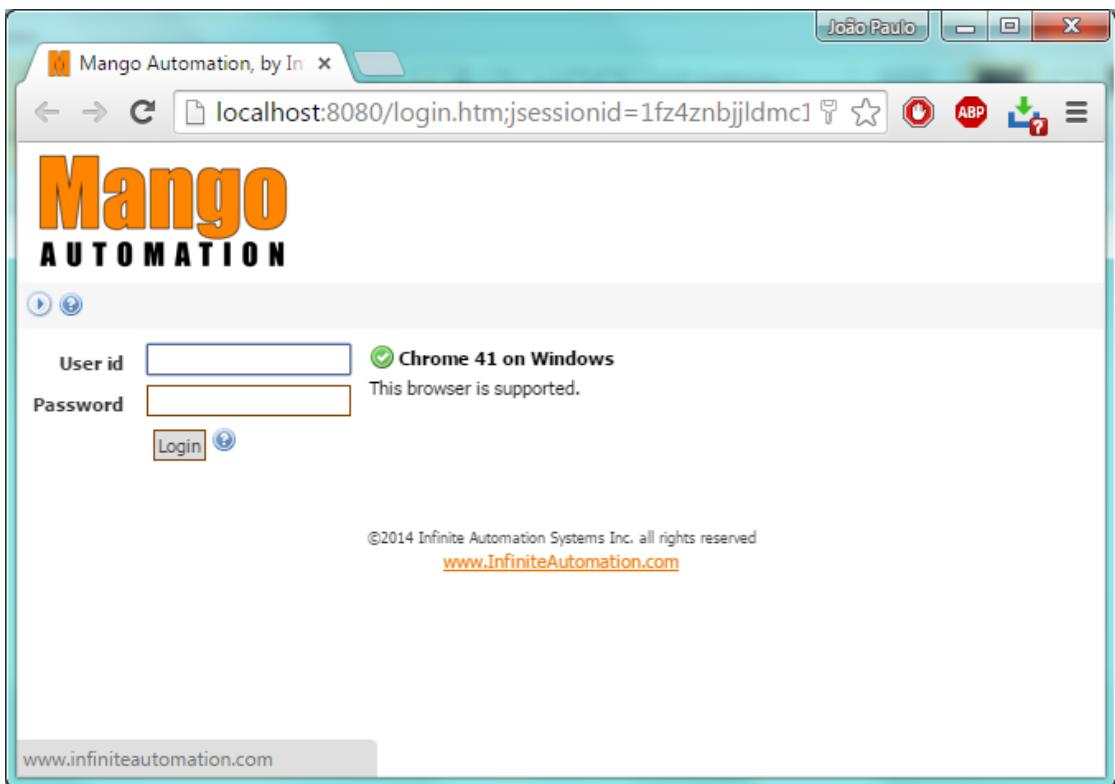


Figura 7.1: Tela de login do mango.

O protocolo modbus é um dos utilizados na automação industrial. Existe uma implementação do modbus serial para arduino, que é implementado pelo uso da biblioteca **SimpleModbusSlave**, que faz o arduino trabalhar como um escravo modbus. O data source Modbus Serial nos permite então obter dados de um arduino, seguindo este protocolo.

A configuração do Modbus para a comunicação com o arduino é tal como mostrado na figura 7.5. É importante que a porta seja a mesma do arduino, bem como bit rate (baud rate), data bits, stop bits, stop bits e parity. O encoding deve ser do tipo RTU e deve se marcar a caixa **Contiguous batches only**.

O modbus trabalha endereçando o escravo pelo *slave id*, que no caso usamos 1. Os dados são armazenados nos chamados registradores (*registers*). Enquanto o Modbus especifica 4 tipos de registradores, o arduino implementa apenas o *Holding register*, logo nossos pontos devem ser todos deste tipo e concordar com a sequencia definida no arduino. Cada registrador é de 16 bits. Podemos definir um ponto binário como sendo um único bit daquele registrador ou um ponto numérico, que usa todos os 16 bits.

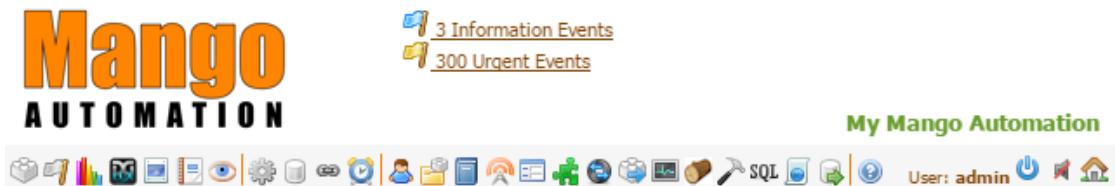


Figura 7.2: Topo da tela do mango.

7.3 *Watch list*

Podemos observar o comportamento dos pontos criados na tela *Watch lists* (figura 7.6). Se o ponto for configurável (settable), também podemos fazé-lo por esta tela.

7.4 Tela gráfica

A interface normal com o operador é feita através do sinótico – a representação gráfica do processo. O mango permite montar uma tela gráfica apresentando os diversos valores das variáveis e os controles, como mostra a figura 7.7. A apresentação de variáveis é de forma bem direta, através dos diversos comandos de gif analógico, gif binário, gif dinâmico e gráficos. O ajuste de valores, porém, é um pouco mais complexo.

Pode-se ajustar o valor de um ponto no mango marcando a opção **Exibir controles**, a partir do que se pode abrir uma caixinha que permite a escrita do valor na variável. Esta opção porém é mutio pouco prática.

Outra opção mais interessante é através dos *server side scripts*, ou scrpts do servidor. Nestes casos define-se um comando em javascript que permite ler e alterar o valor de uma variável. A página <http://infiniteautomation.com/wiki/doku.php?id=graphics> mostra vários exemplos de códigos úteis para esta tarefa.

Um adendo: os códigos da página supracitada consideram os endereços a partir da pasta <mango>/web. Porém o arquivo zipado do Mango não contém aí a subpasta Graphics, o que faz com que os exemplos não funcionem. é necessário copiar a pasta <mango>/web/modules/sstGraphics/web/graphics para este local. Pode-se (e deve-se) acrescentar outras imagens mais interessantes a este diretório.

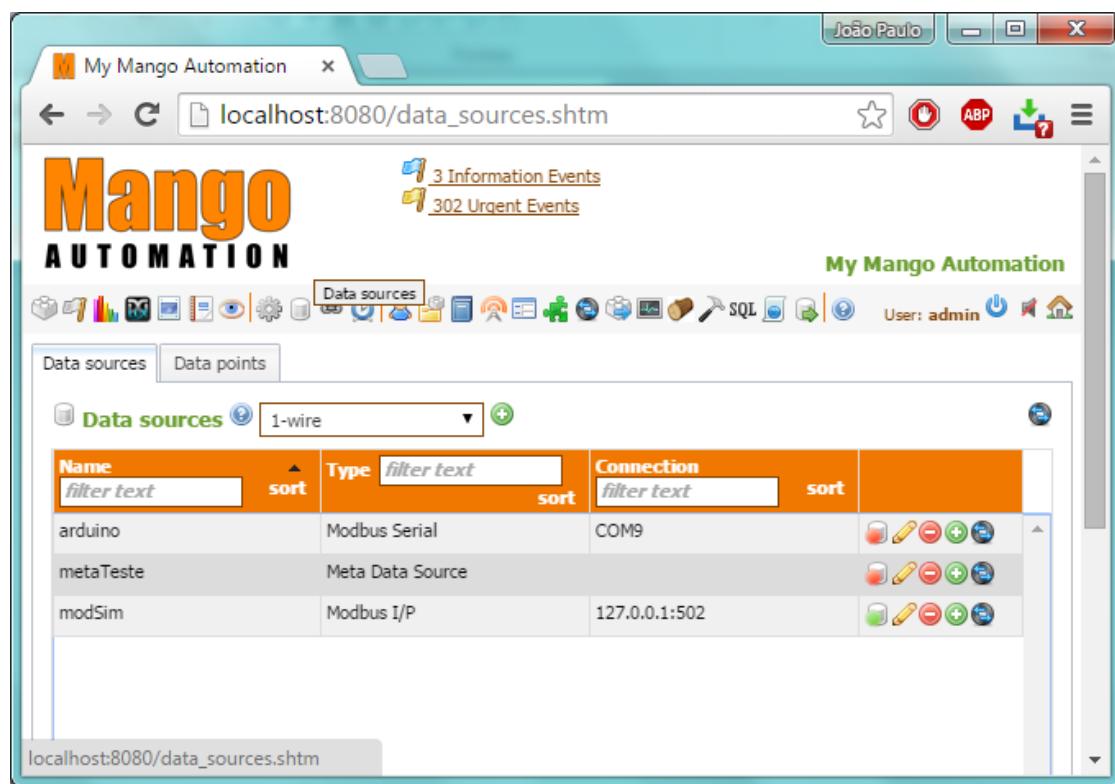


Figura 7.3: Tela de fontes de dados.

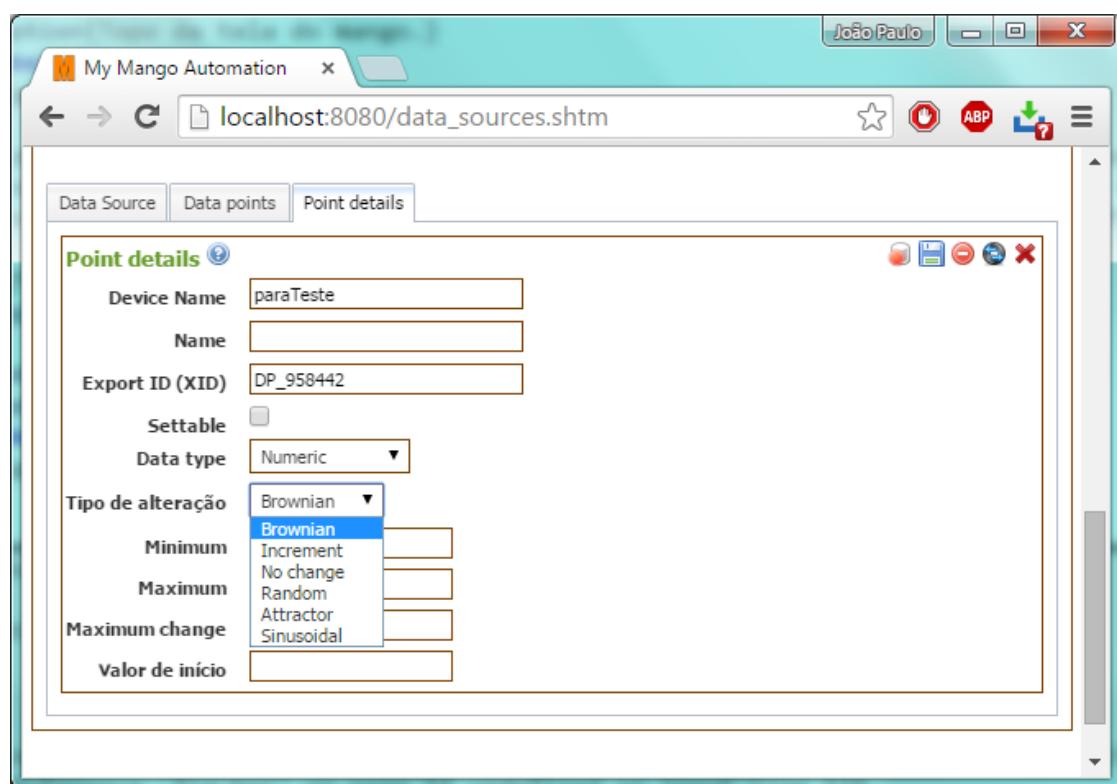


Figura 7.4: Definição de dados virtuais para simulação.

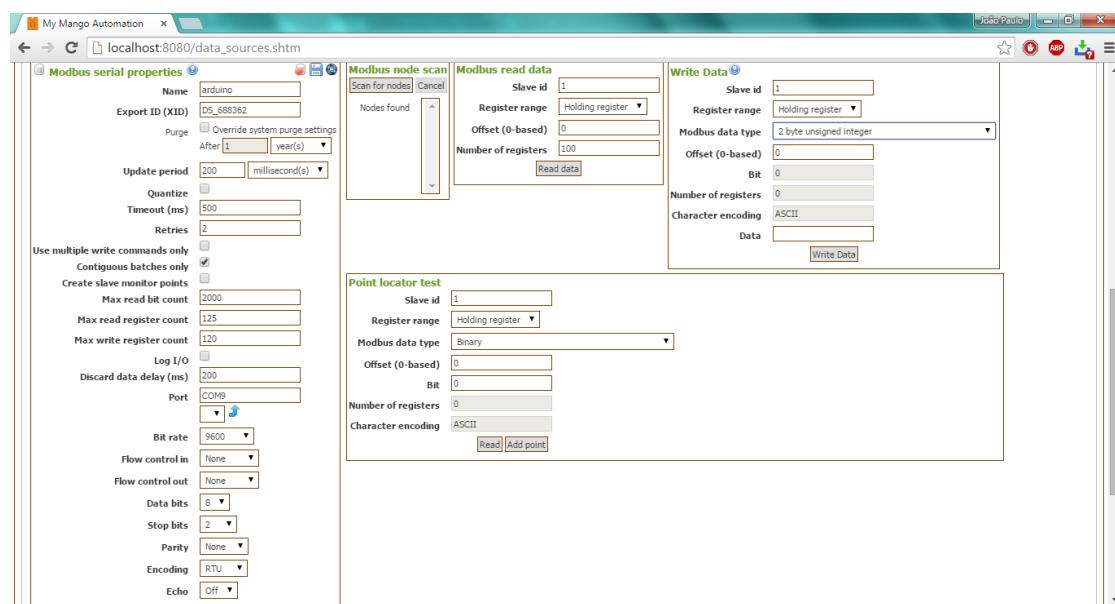


Figura 7.5: Definição de uma fonte de dados do tipo modbus serial.

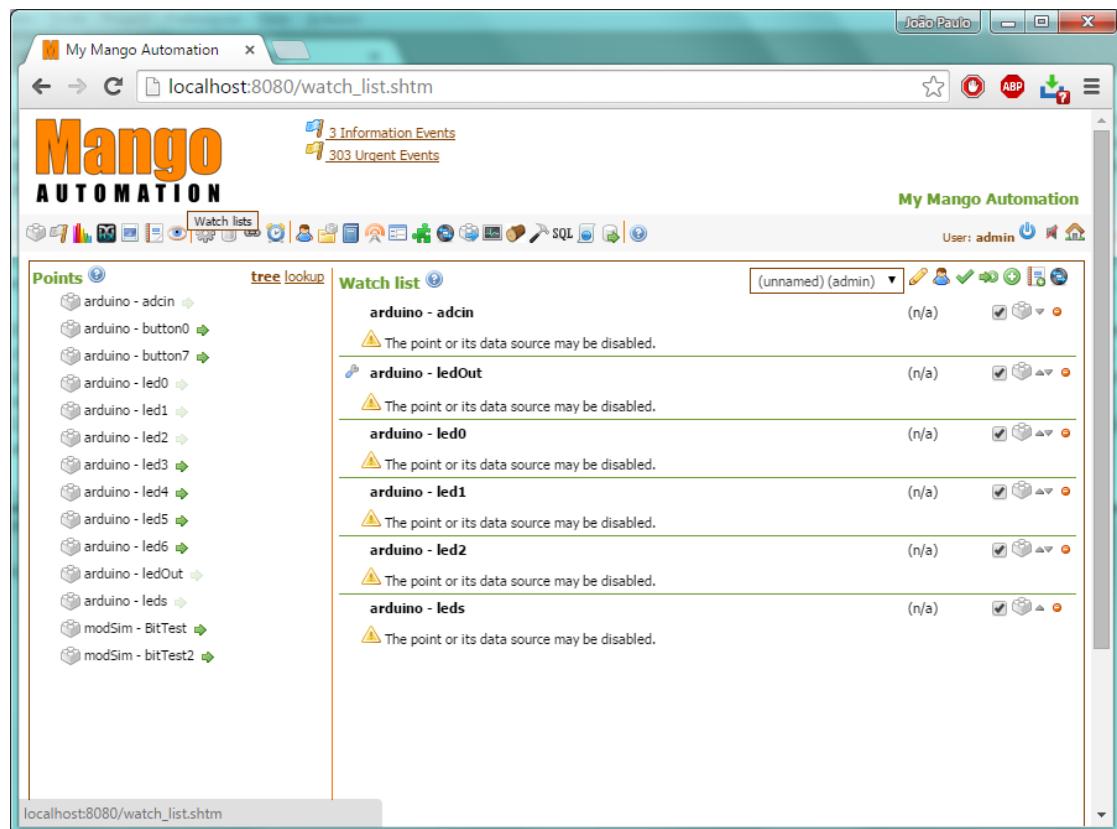


Figura 7.6: *Watch lists* para observar as variáveis.

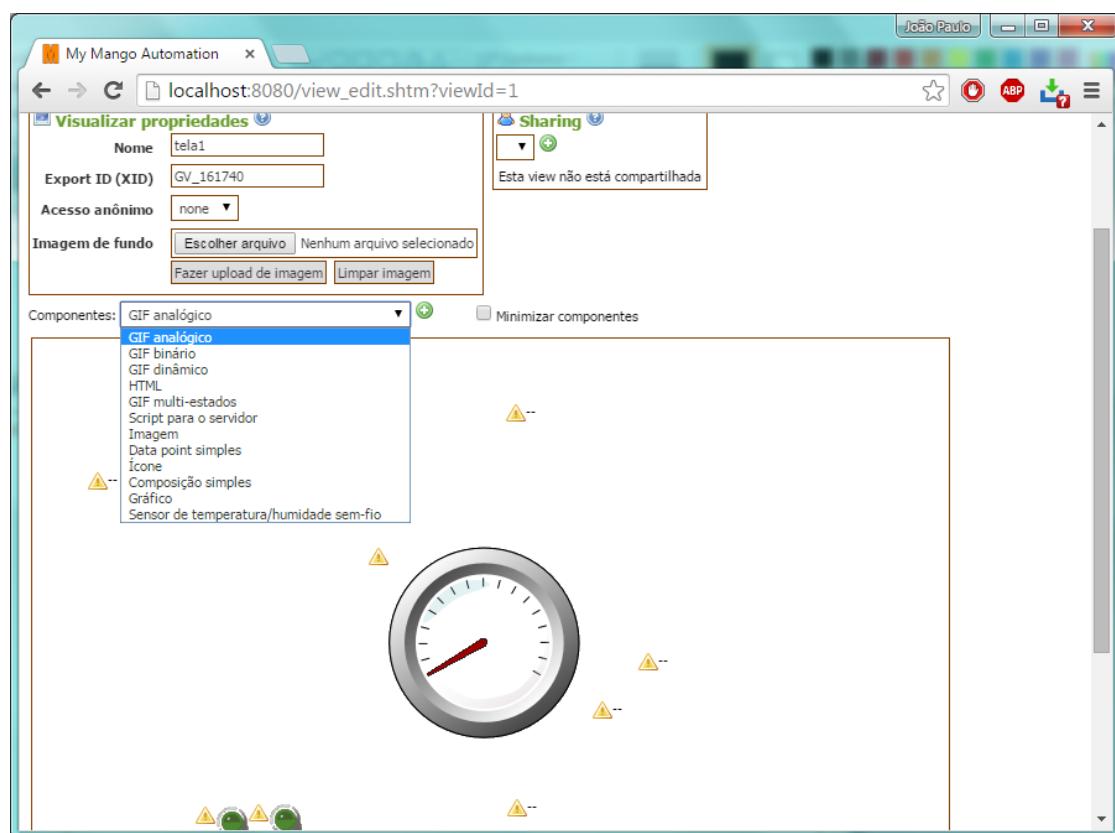


Figura 7.7: Montagem de sinótico.

Capítulo 8

Gerenciamento da manufatura: PIMS e MES

PIMS – *Process Information Management System* e MES – *Manufacturing Execution System* são sistemas da camada 3 da pirâmide de automação, responsáveis pelo armazenamento e tratamento de dados do nível 2, concentrando os dados de diversos processos separados em um único ponto. São os chamados *middleware*, pois ficam a meio caminho entre os sistemas de gerenciamento de empresa e os supervisórios, às vezes combinando funções de um ou de outro.

De forma geral, no terceiro nível da pirâmide a preocupação é em consolidar os dados brutos do processo (*data*), para com eles gerar informações (*information*) e conhecimento (*knowledge*) sobre o processo, aumentando o valor destes valores, como mostra a figura 8.1. Os dados são obtidos ou do controlador ou do supervisório de um determinado processo. A relação entre dados ou a variação destes dados no tempo geram informação sobre a planta. A relação entre informações ou a variação de informações no tempo geram conhecimento.

Um exemplo desta relação é mostrado na figura 8.2. Nesta figura, a partir dos dados de temperatura e vazão de um fluido é gerada a informação do calor removido em determinado trocador de calor. A comparação dos calores removidos de diversos trocadores gera o conhecimento de qual trocador é mais eficiente.

8.1 PIMS

Para a finalidade de gerar informação e conhecimento, o ponto de partida é obter os dados brutos. Esta é a tarefa principal do PIMS: adquirir, armazenar e apresentar diversos dados de uma planta. O PIMS foi criado e ainda é principalmente usado para processos contínuos, tais como uma refinaria ou siderúrgica, e portanto tem um enfoque muito grande em variáveis analógicas e na relação delas com o tempo.

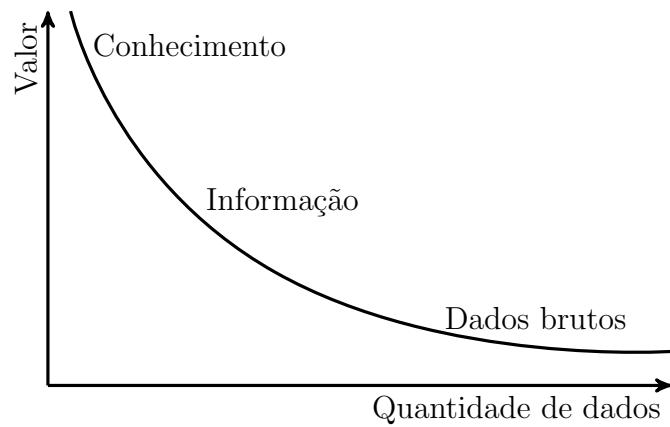


Figura 8.1: Relação entre dados, informações e conhecimentos.

Do ponto de vista do PIMS podemos esquematizar os sistemas de uma fábrica como mostra a figura 8.3, onde se vê que o PIMS tem 4 partes principais: historiador de processos, banco de dados temporal, interface gráfica e aplicações clientes (variadas funções, desde análise dos dados a interface com outros sistemas).

8.1.1 Historiador de Processos

O historiador do processo se comunica com vários sistemas do nível 1 (CLP, CNC) ou 2 (supervisório) ou ainda de outros sistemas nível 3, tais como um LIMS – *Laboratory Information Management System* ou MES para obter dados brutos dos diversos processos e acumula-os no banco de dados. Tal sistema fornece as seguintes funcionalidades:

Registro histórico para análise de incidentes, controle de qualidade, métricas de performance, entre outros.

Adequação a normas como por exemplo para controle ambiental.

Monitoração de equipamentos para controle de vida útil e apoio à manutenção.

Análise de processo facilita a visualização de dados e detecção de correlações.

Os dados coletados são principalmente os valores das variáveis do processo, sejam discretos ou contínuos, mas também abarcam outras informações, tais como a ocorrência de alarmes, a marcação de que operador está presente, o período que um equipamento está ligado, entre outros. Cada uma destas informações é identificada por um marcador único - a chamada *tag*, ao qual também está associado o

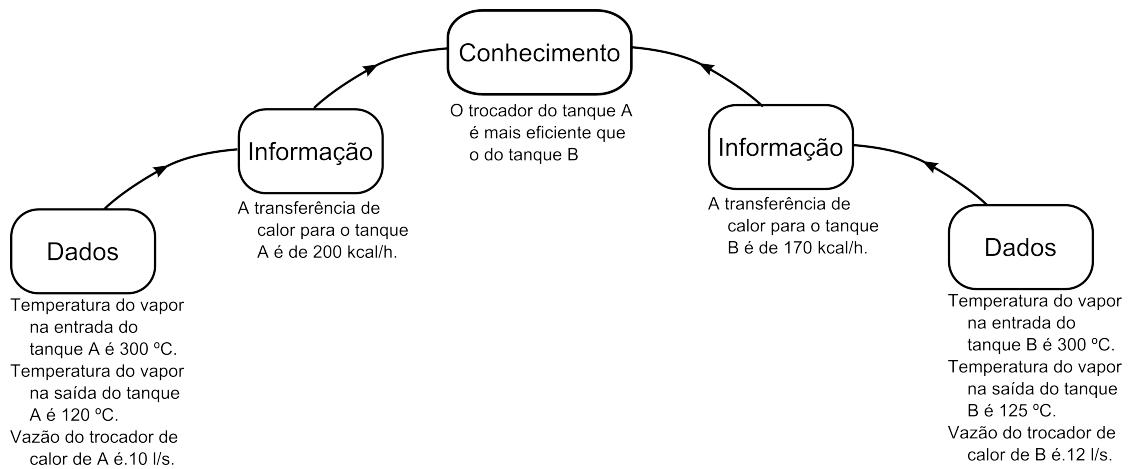


Figura 8.2: Exemplo da transformação de dados para informação e de informação para conhecimento.

endereço lógico de onde se obtém tal informação e o tempo em que tal dado foi gerado (*time stamp*). Em alguns casos se associa também uma métrica da qualidade do dado, referente a confiabilidade daquele dado, tal como se o instrumento de medida está calibrado ou não.

Estas informações podem ser obtidas tanto de sistemas SCADA (nível 2) ou de CLPs (nível 1). Algumas vantagens de pegar informação dos sistemas nível 2 são que:

- o SCADA já converteu os dados para unidades de engenharia enquanto que em alguns CLPs os dados estão em valor bruto (de 0 a 4095);
- muitas variáveis são definidas apenas no sistema SCADA, não existindo nos CLPs, tais como o motivo de alarmes ou qual operador está monitorando a operação;
- interface com os sistemas SCADA costuma ser padrão, o que facilita a comunicação.

Vantagens de obter as informações do CLP são:

- busca dos eventos com menor atraso temporal;
- pode-se coletar os dados em um ponto único, se todas as redes de CLPs estiverem interligadas;
- CLPs são mais confiáveis e apresentam menor suscetibilidade a falhas que os sistemas SCADA.

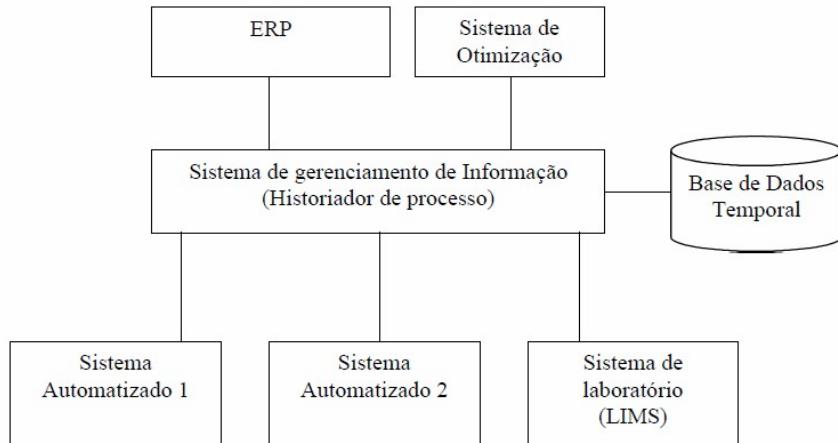


Figura 8.3: Sistema PIMS.

8.1.2 Banco de Dados Temporal

A maioria das análises realizadas nos dados de um sistema PIMS são em função do tempo, logo é comum ele usar um banco de dados que indexa a informação pelo tempo. Basicamente é uma tabela, relacionando *time stamp*, *tag*, tipo de dado (análogico, booleano, texto), valor e qualidade (se houver).

Um problema do uso deste tipo de banco de dados, ao invés dos chamados bancos de dados relacionais, tipo SQL, é que a busca por informação pode ter uma baixa performance quando a quantidade de dados aumenta muito. Esta é a principal razão para que estes sistemas façam uma compressão de dados, que basicamente se resume a não armazenar dados que não tragam muita informação nova. A figura 8.4 mostra a idéia por trás da compressão de dados.

Algoritmos comuns para a compressão de dados no PIMS são *banda morta*, onde os dados são apenas armazenados se variarem mais do que um mínimo especificado; o *SDCA – Swinging Doors Compression Algorithm*, onde para cada valor recebido é definida uma reta entre ele e o último valor armazenado, descartando valores que possam ser definidos por esta reta e mais um erro; e o *boxcar/backslope*, que usa a banda morta e mais uma reta definida pelo último valor armazenado.

8.1.3 Interface Gráfica

A interface gráfica de um sistema PIMS é, em muitos aspectos, muito parecida com a de um sistema SCADA, contendo representações pictóricas do processo (sinóticos) com os valores de várias variáveis e gráficos de tendência. Tais elementos são melhor vistos no contexto de um sistema SCADA.

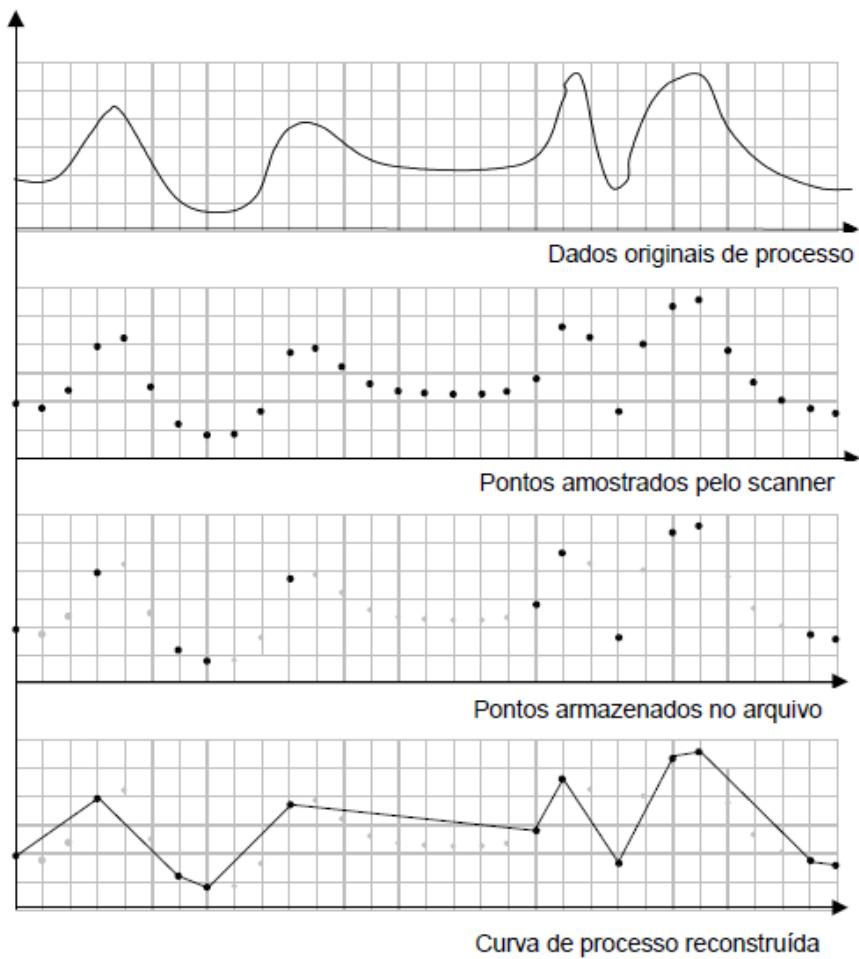


Figura 8.4: Processo de amostragem, compressão e reconstrução dos dados.

8.2 MES

Assim como o PIMS, um sistema MES também adquire dados do processo com o objetivo de apresentar uma visão geral do processo. Porém, enquanto o PIMS é focado mais no armazenamento dos dados, o MES tem uma gama maior de funções e um papel mais ativo. Historicamente, os MES são usados principalmente em processos discretos, muitas vezes em sistemas de automação flexível ou programada. Isto faz com que o MES tenha que lidar com o sequenciamento dos processos, o que ocorre menos em sistemas contínuos.

Os sistemas MES agregam diversas funções de sistemas anteriores mais simples e específicos e são definidos por terem as seguintes funções:

Gerenciamento das definições de produto. Todas informações necessárias para a fabricação do produto. Isto inclue lista de materiais e insumos, set-points do processo e receitas.

Por receita, entenda-se uma variação do processo para, na mesma máquina, produzir produtos diferentes ou variações dele. Por exemplo, num processo de pintura a receita diria quais as tintas usadas, em que sequência, em que volume e em que velocidade de aplicação. É muito importante na automação flexível e programável.

Sistemas de automação fixa também se utilizam de receitas. Podemos citar dois casos mais importantes: para a melhoria do processo e do produto, testando diferentes receitas e checando os resultados e para casos de variações de matéria prima ou de condições ambientais, onde se buscaria a melhor receita para cada caso.

Gerenciamento de insumos. Permite preparar e executar ordens de produção com garantia de disponibilidade dos insumos. Esta função cuida do controle de estoques e dos pedidos aos fornecedores, principalmente quando se usa a metodologia de *just in time*, que minimiza os estoques.

Agendamento de produção. Permite determinar a ordem que a produção será feita, para alcançar os requerimentos de produção definidos pela ERP (camada 4 da pirâmide) utilizando otimamente os recursos. Também chamado de *scheduler*.

A figura 8.5 mostra o exemplo de um *scheduler*, onde se vê o uso de um diagrama de Gantt para definir o sequenciamento das operações e máquinas.

Tipicamente inclui ferramentas de simulação, que permitem comparar diversas opções de ordens e estimar efeitos quando de mudanças imprevistas na sequência de produção (em geral por conta de uma parada não programada).

Envio de ordens de produção. Em função do agendamento feito, o MES cuida de enviar as ordens de produção para os diversos postos da planta. A informação vai para os supervisórios e, em alguns casos mais avançados, para os controladores.

Acompanhamento da execução de ordens de produção. Também se comunica com sistemas níveis 1 e 2 para garantir a execução das ordens. Inclui também o registro de paradas.

O registro de paradas é feito automaticamente quando o equipamento para, seja por alguma condição espúria detectada no nível 1 ou por ação do operador no nível 2. Em ambos os casos fica registrado uma parada em aberto, que

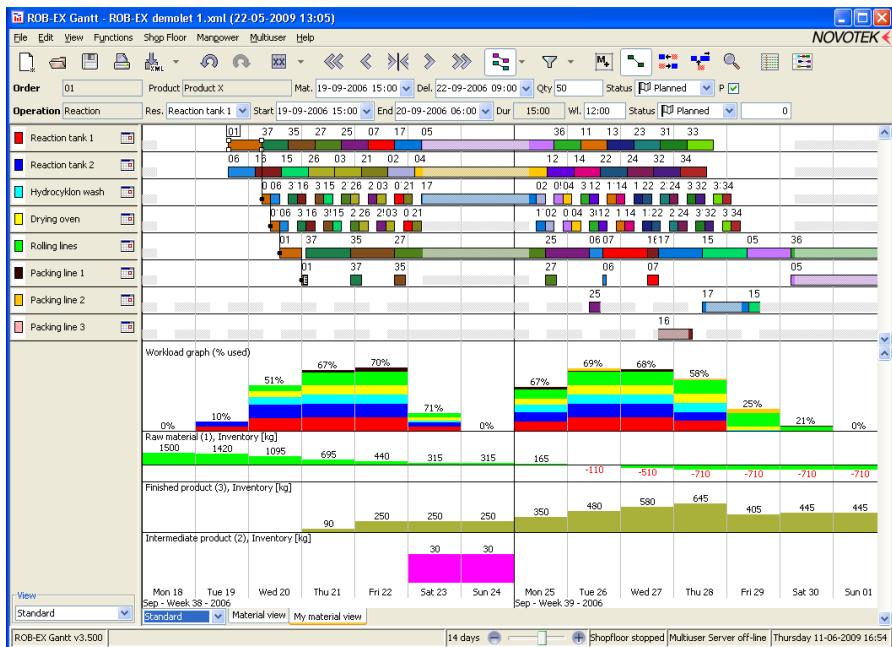


Figura 8.5: Exemplo de um *scheduler*.

só finaliza quando o operador complementar certas informações que auxiliam no diagnóstico da parada e o equipamento voltar a funcionar.

Coleção dos dados de produção. Equivalente ao historiador de processo do PIMS.

Análise da performance da produção. Cálculo dos chamados índices de produção – *KPI*, *Key Performance Indicators*, tais quais na figura 8.6. É a geração de informação a partir dos dados da produção.

O OEE – *Overall Equipment Effectiveness* é um exemplo de KPI não diretamente ligado ao produto, mas à produção. O OEE aponta a efetividade de um determinado equipamento ou célula de produção. Este índice é dado por:

$$\text{OEE} = \text{disponibilidade} \times \text{performance} \times \text{qualidade}, \quad (8.1)$$

onde por disponibilidade entende-se a razão entre o quanto de tempo o equipamento funcionou e o quanto de tempo ele deveria ter funcionado, ou seja, descontando-se as paradas não programadas; performance é a razão entre a produção do equipamento enquanto funcionava e a capacidade de produção

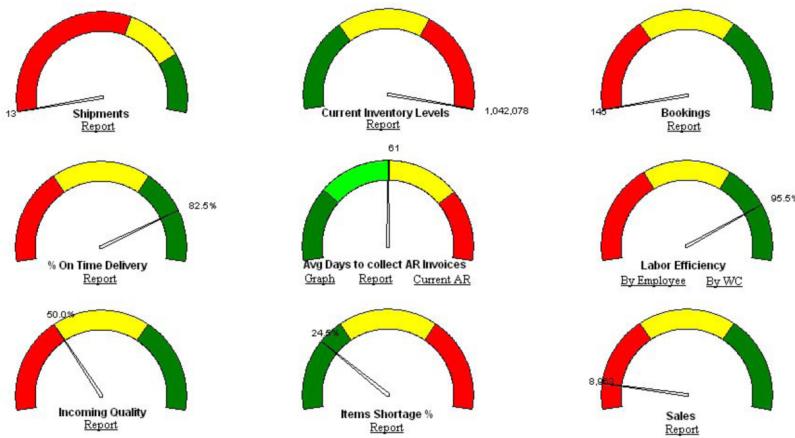


Figura 8.6: Exemplo de visualização de KPIs.

de que o equipamento é capaz, ou seja, descontando a ociosidade do equipamento; e qualidade é o valor do que o equipamento produziu em relação ao valor se não tivesse havido nenhum descarte ou geração de produtos de menor valor. Esta relação é melhor visualizada na figura 8.7 e exemplificada na figura 8.8.

■ **OEE = Disponibilidade * Performance * Qualidade**

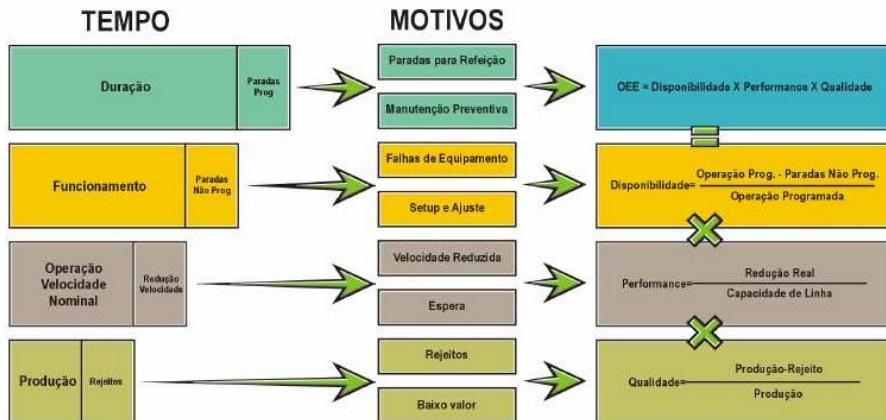


Figura 8.7: Overall Equipment Effectiveness.

Rastreamento da produção. Permite levantar que produto ou lote foi feito quando e em qual equipamento. Útil para melhoria da produção e imprescindível para remédios e produtos alimentícios.

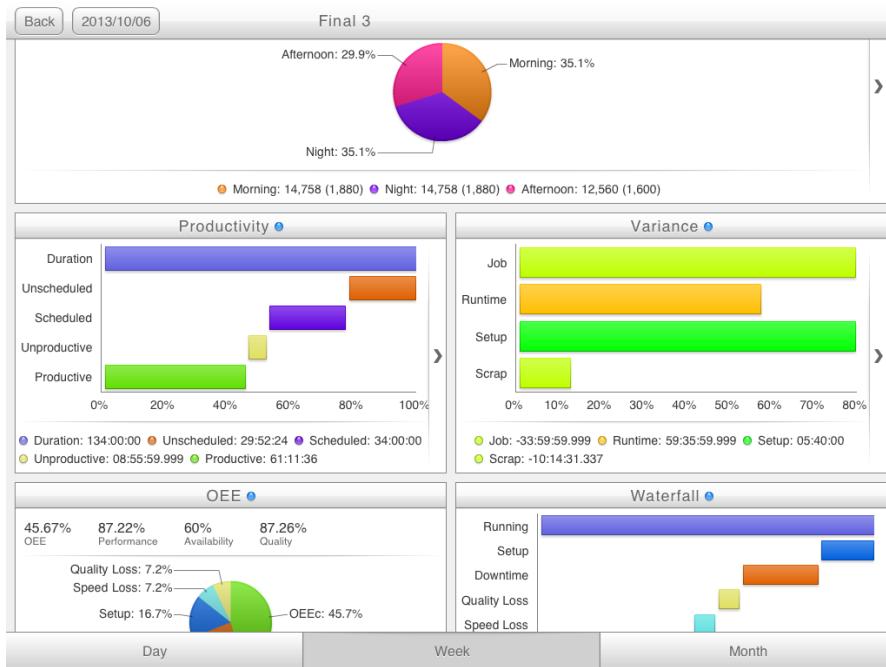


Figura 8.8: Overall Equipment Effectiveness.

Armazenamento dos logs de produção. Hoje em dia tais logs são inseridos pelo operador no próprio sistema supervisório e realcionados às variáveis de produção.

Interface de auditoria. Permite a análise dos diversos dados e informações armazenados e o cruzamento destes dados com outras bases de dados.

8.2.1 Redes industriais

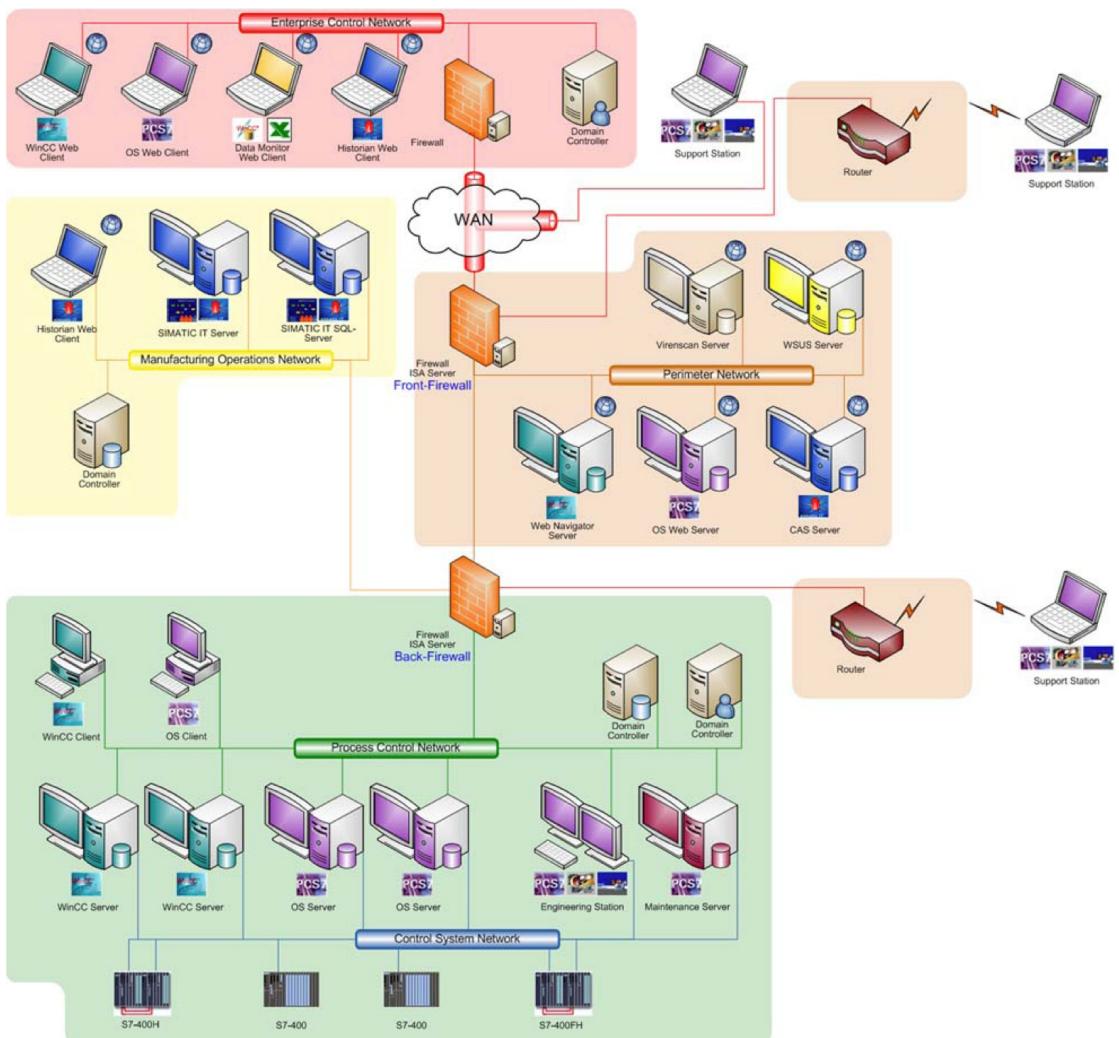


Figura 8.9: Arquitetura de rede de dados industrial.

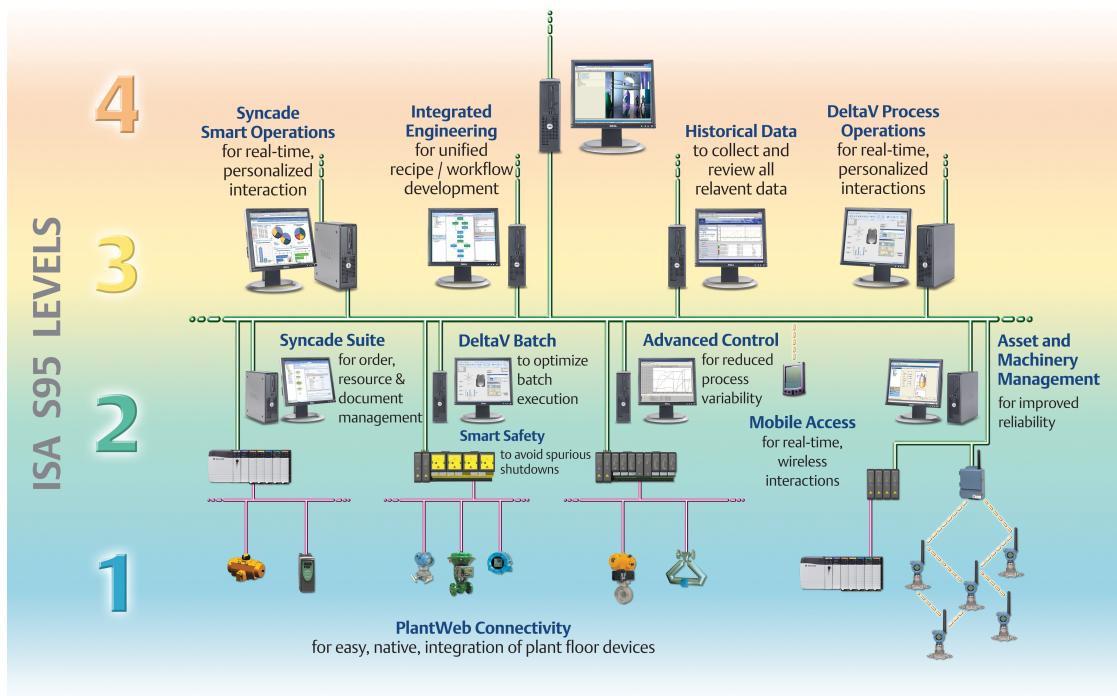


Figura 8.10: Arquitetura de rede de dados industrial.

8.3 Gerenciamento da manufatura: MES

8.4 Redes de Comunicação: Introdução e noções básicas

Na pirâmide de automação, a partir da camada 2, os sistemas utilizados são softwares em computadores, sejam servidores, terminais, desktops, notebooks ou mesmo tablets e smartphones (para o caso do supervisório). No nível de controle, o dispositivo utilizado pode ser um computador, um CLP – que não deixa de ser um computador – ou outro dispositivo que incorpora um processador. Hoje em dia até no nível 0 estão ficando cada vez mais comuns os dispositivos ditos *inteligentes*, que incorporam também um microprocessador. A comunicação entre eles é feita por uma rede de dados, tais como a ethernet.

O problema da comunicação de dados automática entre diversos dispositivos é bastante complexo. Ele envolve a:

1. transferência de dados de um dispositivo a outro,
2. em alta velocidade,
3. sem perda de informação,
4. eventualmente passando por dispositivos intermediários,
5. talvez envolvendo sistemas de comunicação diferentes,
6. com privacidade,
7. sem envolver outros dispositivos desnecessariamente.

Peguemos como exemplo pagar uma conta via internet. Logo é necessária a comunicação do dispositivo caseiro (que seja um tablet) e o computador central do banco (1). Quanto mais rápida for esta comunicação, mais agradável é para o usuário e mais barato é para o banco (2). Obviamente nenhuma das partes envolvidas querem que seja pago o valor errado ou a conta errada (3). Entre o tablet e o servidor do banco estão: o modem wifi, a central telefônica, servidores da companhia telefônica e talvez de outras fornecedoras de serviço (4), e em geral não é do interesse do usuário que a companhia telefônica ou outro elemento saiba de sua senha do banco (6). A comunicação do tablet ao modem é por wifi. Do modem para a central telefônica é pelo fio telefônico. Da central em diante pode ser por cabos de cobre ou, mais provavelmente, por fibra óptica (5). Também, por questão de custos e privacidade, não é desejável que outros elementos recebam aquela informação, mesmo que criptografada.

Para resolver este problema, foi desenvolvida a idéia de quebrá-lo em várias partes, ou camadas, onde cada camada é responsável por uma determinada parte

do problema. A informação fica então passando de camada a camada para resolver todas as questões relativas à comunicação. Tal esquema é também chamado de pilha de protocolos.

8.4.1 Modelo OSI

Diferentes tipos de rede de dados tem problemas diferentes. O modelo OSI procura definir camadas o mais genéricas possíveis, de modo a abranger qualquer tipo de comunicação de dados. Este modelo define 7 camadas: física, enlace, rede, transporte, sessão, apresentação e aplicação.

Física

A camada física é a única que não pode ser implementada em software. Ela lida com as definições eletro-mecânicas necessárias para a comunicação. Ou seja, a camada física define:

- a forma como os dados são transmitidos (sinais elétricos por cabo, sinais de rádio, luz, etc);
- as características do meio de transmissão, tais como o tipo de cabo, os níveis de tensão, o formato dos conectores, etc;
- a taxa de transmissão.

Note-se que a taxa de transmissão acaba sendo limitada justamente pelos diversos parâmetros físicos descritos pela camada física de um determinado protocolo de comunicação.

Outro parâmetro definido na camada física é se a rede é *simplex*, *half-duplex* ou *full-duplex*. Simplex significa que a comunicação no meio de transmissão é apenas num sentido (exemplo: televisão), half-duplex é uma comunicação em dois sentidos, mas apenas um por vez (exemplo: walkie-talkie) e full-duplex é uma comunicação em ambos os sentidos ao mesmo tempo (exemplo: telefonia).

Enlace

A camada de enlace cuida da comunicação de um dispositivo a seu vizinho, que não necessariamente são os pontos finais.

No exemplo acima, esta seria a comunicação entre o tablet e o modem wifi; entre o modem e a central; entre a central e o servidor da companhia telefônica, e assim por diante.

A camada de enlace de um protocolo define quem acessa o meio de transmissão e quando, que sinais indicam o início e o fim de uma transmissão, mecanismos que

detectem e/ou corrijam erros e, se necessário, para qual dentre vários dispositivos é aquela comunicação específica.

Alguns protocolos de comunicação, tais como USB e ethernet, fazem a conexão ponto a ponto. Neste caso apenas 2 dispositivos podem acessar o meio. Outros sistemas, como o wifi, podem ter vários (em alguns casos, centenas) de dispositivos no mesmo meio. O que torna necessário a definição de quem pode acessar o meio em cada instante. Várias possibilidades existem:

mestre-escravo Neste sistema, a comunicação sempre é iniciada pelo mestre, e um escravo só ocupa o meio quando o mestre requisita uma informação. Ex.: USB, Modbus.

token ring É passada uma ficha (token) virtual entre os dispositivos. Quem tem a ficha pode falar.

Multiplexação por tempo Cada dispositivo tem uma hora específica para controlar o barramento. Um sinal periódico (*NUT- Network Update Time*) sincroniza os dispositivos.

CSMA-CD – *Carrier Sense Multiple Access with Collision Detection*
Sempre que um dispositivo quer se comunicar com outro, ele checa antes se o meio está livre. Ex.: wi-fi, bluetooth, CAN.

Tipicamente os protocolos da camada de enlace acrescentam uma certa redundância ao dado transmitido, o que permite detectar e até corrigir erros de transmissão. São os chamados códigos corretores de erro, dentre os quais se utiliza bastante o CRC - *Cyclic Redundancy Check*.

Rede

Camada responsável pelo roteamento da informação. Ou seja: Se A quer se comunicar com D, mas A apenas tem ligação direta com B e C, para quem A deve mandar a informação?

Tipicamente este roteamento é feito através de uma tabela que mapeia o endereço lógico de um dispositivo (o endereço usado nas etapas acima, tais como o IP) e o endereço físico, usado pela camada de enlace (muitos sistemas usam o *MAC address*). Esta é a última camada a que uma comunicação chega num dispositivo que não o inicial ou final: o programa desta camada checa se o endereço lógico da mensagem é o mesmo daquele dispositivo; não sendo, ele busca na tabela qual o endereço físico para o qual reenviar aquela mensagem e reenvia, tal qual mostra a figura 8.11. Isto serve também para fazer a conexão entre diferentes redes.

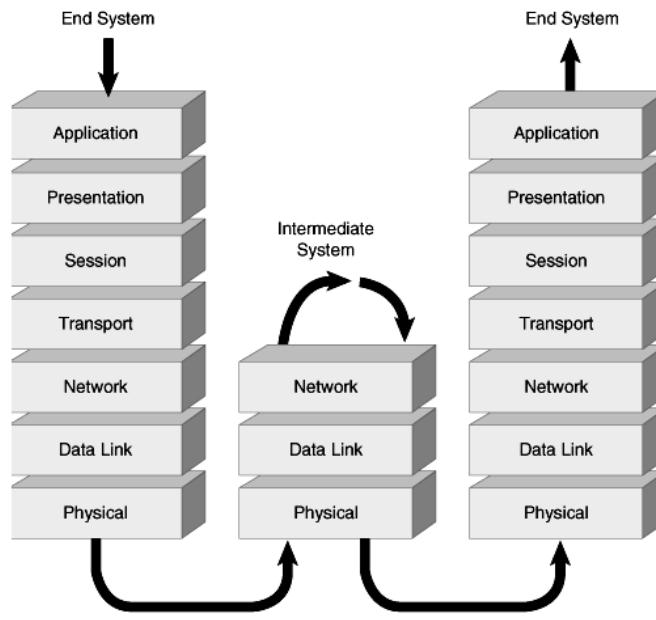


Figura 8.11: Comunicação passando por dispositivo intermediário.

Transporte

A camada de transporte cuida da comunicação entre o ponto inicial e final. Usa apenas o endereço lógico. Esta camada pode resolver quebrar uma mensagem grande em vários pacotes, mandando um pacote de cada vez para a camada de rede e remontando a mensagem no recebimento.

Um conceito interessante utilizado na camada de transporte é o de *Quality of Service – QoS*, qualidade de serviço. Um sistema com alto QoS garante a chegada de todos os pacotes, na ordem em que foram enviados.

A internet foi originalmente pensada como um sistema de baixo QoS, voltado para a transmissão de arquivos. A ordem dos pacotes não interessa e nem o atraso de um pacote. Caso um pacote se perca, simplesmente pede-se que seja reenviado. Uma rede de telefonia, por outro lado, busca garantir que o tempo de cada pacote seja o mesmo, para não piorar a qualidade da voz transmitida.

Para diversas aplicações industriais, o QoS é importante: não se pode aceitar que uma mensagem de alarme, por exemplo, demore muito a chegar. EtherCAT - *Ethernet for Control Automation Technology* é uma modificação de Ethernet para mensagens com diferentes QoS.

Sessão

Esta camada cria, gerencia e termina conexões entre 2 pontos de uma rede. É mais comum na telefonia, onde se gera um caminho fixo para uma determinada ligação. Basicamente esta camada gera a tabela que é usada pela camada de rede.

Apresentação

Faz a mudança no formato dos dados. Exemplo: troca de quebra de linha entre sistemas Unix e Windows, mudança de codificação windows 1252 para UTF, etc.

Inclui também a criptografia, que é a base para garantir a privacidade da comunicação ponto-a-ponto.

Aplicação

A aplicação final

8.4.2 Internet

A Internet define uma pilha de apenas 4 protocolos: o de enlace, que engloba também o físico; o IP – *Internet Protocol*, equivalente ao de rede do modelo OSI; o de transporte, que pode ser um entre vários, sendo o TCP e o UDP os mais comuns e o de aplicação, que envolve sessão, apresentação e aplicação num só.

A figura 8.12 mostra a sequência que é feita com uma informação passada por http pela internet.

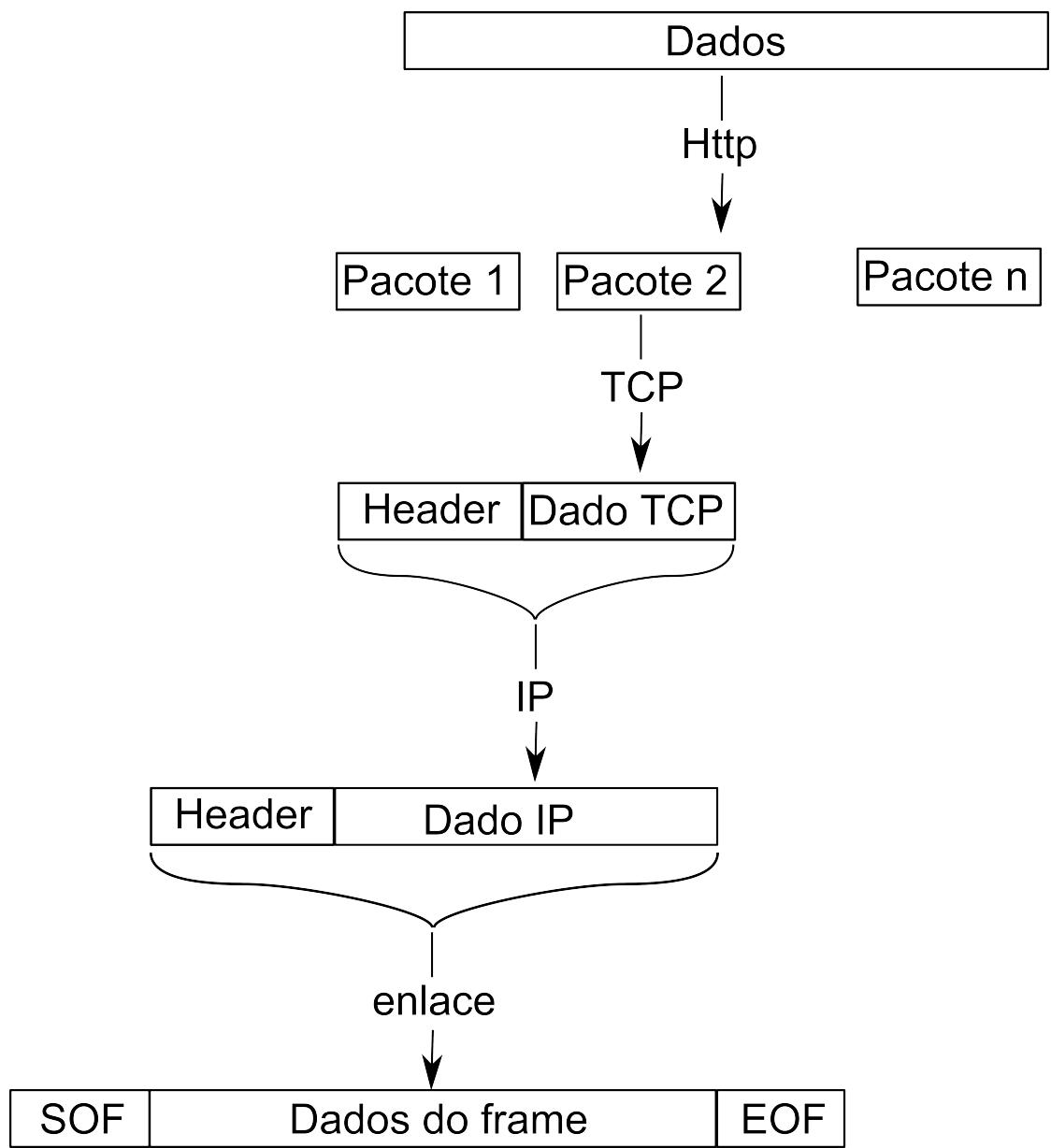


Figura 8.12: Encapsulamento da informação na pilha TCP/IP