Capítulo 5

Aplicação das Abordagens Assíncrona e Síncrona

O objetivo deste capítulo é mostrar, através de exemplos mais significativos, como as técnicas apresentadas ao longo deste livro podem ser aplicadas na prática. A seção 5.1 trata de uma aplicação do tipo "veículo com navegação autônoma", a qual será tratada segundo a abordagem assíncrona, isto é, utilizando a análise de escalonabilidade, um sistema operacional de tempo real e uma linguagem de programação como C ou C++. A seção 5.2 apresenta uma aplicação do tipo "sistema de controle", a qual em função das suas características, será tratada segundo a abordagem síncrona, utilizando a linguagem Esterel. A seção 5.3 contém uma discussão sobre as duas abordagens e elementos para a comparação e a melhor utilização destas.

É importante observar que a limitação de espaço impossibilita uma descrição completa de tais aplicações. Entretanto, mesmo com a descrição simplificada apresentada aqui pretende-se permitir ao leitor uma melhor compreensão de como as técnicas apresentadas neste livro podem ser usadas em aplicações reais.

5.1 Aplicação com Abordagem Assíncrona

Esta seção descreve um sistema com requisitos de tempo real, o qual será utilizado para exemplificar a aplicação das técnicas apresentadas nos capítulos 2 e 3 deste livro. O sistema descrito consiste de um veículo com navegação autônoma (AGV – *"Automatically Guided Vehicle"*) bastante simplificado. O objetivo não é descrever o projeto completo de tal aplicação (tarefa para vários livros do tamanho deste), mas sim ilustrar aquilo que foi discutido. A literatura sobre o tema é vasta. Por exemplo, em [POS97] e [BCH95] que inspiraram o problema a ser tratado a seguir, são descritas soluções completas para um veículo submarino e um veículo para realizar inspeções em depósitos de lixo tóxico, respectivamente.

5.1.1 Descrição do Problema

Um veículo com navegação autônoma deve ser capaz de planejar e executar a tarefa especificada. Veículos com navegação autônoma são usados, por exemplo, para

transporte de material no chão da fábrica, para a inspeção em áreas de risco ou depósitos de material tóxico. Eles podem ser usados como trator para puxar algo, no transporte de material ou ainda como base móvel para um equipamento. Após a determinação do plano a ser seguido, sua execução é iniciada. Um sistema de sensores é usado para dirigir o veículo. Os componentes básicos de um AGV incluem:

- Partes mecânicas e sistema de propulsão;
- Sistemas eletrônicos;
- Sistema sensorial, com sensores internos e externos;
- Módulo de planejamento e navegação;
- Representação interna do mundo em volta do veículo.

O AGV considerado será usado para o transporte de material. Ele tipicamente desloca-se até o local onde deverá pegar o material a ser transportado (navegação), posiciona-se de forma a realizar a carga do material (atracagem), coloca o material em seu compartimento de carga (carga), manobra para sair do local de carregamento (desatracagem), desloca-se até o local destino da carga (navegação), manobra para entrar no local de descarga (atracagem) e finalmente coloca no material transportado no seu destino (descarga). É possível observar que existem 3 fases distintas na operação do AGV, as quais resultam em 3 modos de operação independentes:

- Carregamento e descarregamento de material;
- Planejamento da rota e navegação;
- Manobra para atracar e desatracar de algum tipo de estacionamento.

Com respeito ao modo de operação "planejamento da rota e navegação", é possível dividir o trabalho a ser feito em 3 níveis [RNS93]:

- Nível de planejamento;
- Nível de navegação;
- Nível de pilotagem.

Neste livro vamos considerar exclusivamente o nível de navegação. A navegação parte do plano original desenvolvido pelo nível de planejamento e, considerando as condições locais informadas pelos sensores, dirige o veículo. Diferentes tipos de sensores podem ser usados, tais como bússula, medidor de inclinação, contadores de pulsos nas rodas, cameras de vídeo, sensores de proximidade, sensores de contato, etc. A navegação pode ainda ser auxiliada por emissores de infra-vermelho ou rádio-freqüência colocados ao longo da fábrica.

Desvios da rota inicialmente traçada podem ser necessários em função de obstáculos e para evitar colisões. Quanto maior o número de objetos móveis no chão da fábrica mais complexa torna-se a navegação. No caso da rota original tornar-se inviável, em função de um obstáculo que não pode ser contornado, uma nova etapa de planejamento torna-se necessária. O objetivo da navegação é determinar a direção e velocidade desejada para o veículo, a qual é passada para o nível de pilotagem, responsável pelas operações de controle elementares. A pilotagem utiliza laços de

controle para garantir a correta implementação das decisões de direção e velocidade recebidas do nível de navegação.

No projeto em questão a navegação utiliza uma combinação de três informações: monitoração do movimento das rodas, observação de pontos de referência no ambiente e conhecimento prévio de um mapa básico do ambiente. O movimento das rodas é medido e marcado sobre o mapa do ambiente, o qual foi carregado previamente. A estimativa da posição do veículo a partir do movimento das rodas torna-se mais imprecisa na medida que a distância percorrida aumenta. O erro retorna para zero quando a observação de pontos de referência no ambiente permite determinar com exatidão a posição do veículo. Além disto, obstáculos podem ser observados e, neste caso, o mapa do ambiente é atualizado. A partir da posição, velocidade e direção atuais do veículo, e do plano a ser seguido, são definidas a nova direção e velocidade a serem implementadas. A figura 5.1 apresenta em forma de diagrama de fluxo de dados as funções do nível de navegação.

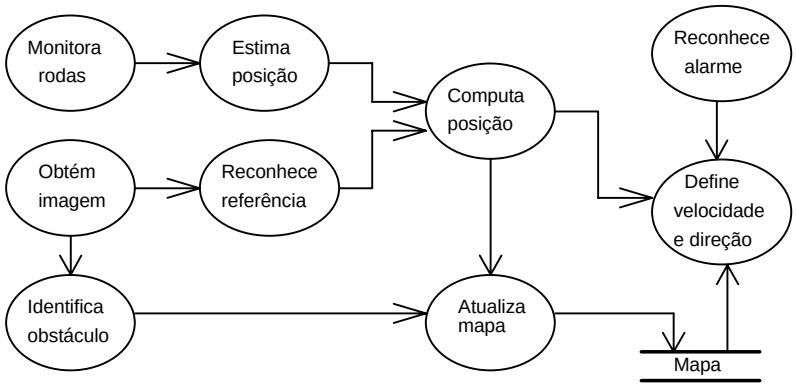

Figura 5.1 - Diagrama do nível de navegação.

A física do veículo impõe restrições temporais para a navegação, para que esta seja capaz de evitar colisões e realizar as manobras com segurança. A definição de velocidade e direção tem o período definido pela física do veículo, sua velocidade máxima, caraterísticas do terreno, etc. Neste caso os engenheiros de controle definiram 100ms como período máximo. A estimativa da posição a partir da monitoração das rodas deve ser feita a cada 100 ms. O reconhecimento de referências não necessita ser tão freqüente, até porque o algoritmo usado é complexo. Uma tentativa de reconhecimento de referências a cada 1300 milisegundos é considerado satisfatório. Por outro lado, a identificação de obstáculos deve ser feita a cada 500 ms. Esta freqüência permite que o veículo, mesmo em velocidade máxima, desvie de um obstáculo que surge repentinamente, por exemplo um outro veículo. Além das funções normais, um auto-diagnóstico deve ser executado no prazo de 20 ms sempre que um sinal de comando for recebido, por exemplo, de uma estação de supervisão.

No caso do AGV descrito, os períodos de funções que não foram definidos explicitamente pela especificação podem ser facilmente deduzidos. Como obstáculos são identificados a partir das imagens, a obtenção de imagens também deve ser feita a cada 500ms. Como a atualização do mapa acontece em função da identificação de obstáculos, ela também deverá ser executada a cada 500ms.

5.1.2 Definição das Tarefas

Uma vez especificado o comportamento temporal desejado conforme descrição anterior, o projetista de software define as tarefas e os tempos associados a partir da especificação, conforme a função de cada uma. O fato de duas tarefas trocarem mensagens ou acessarem estruturas de dados compartilhadas podem criar situações de precedência ou exclusão mútua. Finalmente, a codificação das tarefas e o processador escolhido vão definir os tempos máximos de execução de cada tarefa. Em resumo, temos tipicamente que:

- Períodos e *"deadlines"* são definidos pelos algoritmos de controle empregados;
- Relações de precedência e exclusão mútua são definidos pelo projeto do software;
- Tempo máximo de execução é definido pela programação das tarefas e escolha do processador.

Neste livro, adota-se valores arbitrários para os tempos de execução das tarefas. Entretanto, estes tempos assim como as demais características das tarefas, foram em parte baseadas em uma descrição semelhante encontrada em [But97].

A partir de um estudo cuidadoso da especificação é possível perceber que todas as funções são periódicas com deadline igual ao período, com exceção do tratamento de sinais de alarme, os quais podem acontecer a qualquer instante. Serão empregadas 7 tarefas de software para implementar o nível de navegação do AGV. É suposto que elas devem executar no mesmo processador.

- COMPUTA_POSIÇÃO (C_P)

Tarefa periódica que lê os sensores acoplados às rodas do veículo e estima a posição atual do veículo, em função da posição anterior e do comportamento das rodas, com período de 100ms e tempo máximo de execução de 20ms. Se alguma referência tiver sido reconhecida desde a última execução desta tarefa, então esta informação é também utilizada para computar a posição atual do veículo.

- LÊ_IMAGEM (L_I)

Tarefa periódica que lê a imagem gerada por uma câmara CCD, trata a imagem e armazena em uma estrutura de dados. Período de 500ms e tempo máximo de execução de 20ms. A imagem gerada é colocada em uma estrutura de dados do tipo *"buffer*

duplo", de forma a não criar situações de bloqueio para as tarefas que consomem esta informação.

- ATUALIZA_MAPA (A_M)

Tarefa que usa a imagem gerada pela tarefa LÊ_IMAGEM e procura identificar obstáculos. Em caso positivo, o mapa do ambiente mantido pelo veículo é atualizado. Período de 500ms e tempo máximo de execução de 100ms. Existe uma relação de precedência direta entre LÊ_IMAGEM e ATUALIZA_MAPA.

- RECONHECE_REFERÊNCIA (R_R)

Tarefa que usa a imagem mais recentemente gerada pela tarefa LÊ_IMAGEM e procura reconhecer referências. Em caso positivo, armazena a informação em estrutura de dados a ser consumida pela tarefa COMPUTA_POSIÇÃO, sem entretanto estabelecer uma relação de precedência. Período de 1300ms e tempo máximo de execução de 200ms.

- DEFINE_VELOCIDADE_DIREÇÃO (D_V_D)

Tarefa que usa a posição atual computada e a versão mais recente do mapa do ambiente para definir a velocidade e direção que o veículo deve assumir. Estas informações são passadas para o nível de pilotagem, responsável pela sua implementação. Esta tarefa possui período de 100ms e tempo máximo de execução de 30ms. Existe uma relação de precedência entre a tarefa COMPUTA_POSIÇÃO e a tarefa DEFINE_VELOCIDADE_DIREÇÃO.

- EXECUTA_DIAGNÓSTICO (E_D)

Tarefa aperiódica disparada por um sinal de rádio, ela executa um autodiagnóstico sobre o sistema, sendo capaz de parar o veículo de maneira controlada se uma situação de emergência for detectada. Possui um deadline de 20 ms e é suposto que o intervalo mínimo entre ativações é de 2 segundos. O tempo máximo de execução desta tarefa é 1 ms.

- RELATA (R)

Tarefa que informa a estação de supervisão sobre sua posição, velocidade e direção, a partir dados obtidos pelos sensores. Esta tarefa não é crítica (*"soft"*), pois não afeta o deslocamento e o comportamento do veículo.

Existe ainda uma relação de exclusão mútua entre as tarefas RECONHECE_REFERÊNCIA e COMPUTA_POSIÇÃO, no momento que esta última acessa as informações sobre referências identificadas. O tempo máximo de execução da seção crítica neste caso é de 1ms. Também existe uma relação de exclusão mútua entre as tarefas ATUALIZA_MAPA e DEFINE_VELOCIDADE_DIREÇÃO em função do mapa do mundo. O tempo máximo de bloqueio neste caso é de 3 ms.

Uma vez definidas as tarefas e suas características temporais, é necessário avaliar a

escalonabilidade do sistema, o que será feito na próxima seção.

5.1.3 Modelo de Tarefas

A tabela 5.1 resume a definição das tarefas, como descrito na seção anterior. Antes de proceder à análise de escalonabilidade, é necessário adaptar este conjunto de tarefas no sentido de refletir também os custos do sistema (*"overhead"*) e outros aspectos relativos à implementação.

Tarefa	Tipo	P (ms)	D (ms)	C (ms)	Predecessor	Bloqueio (ms)
C_P	Periódica	100	100	20		1 (c/R_R)
L_I	Periódica	500	500	20		
A_M	Periódica	500	500	100	L_I	3 (c/D_V_D)
R_R	Periódica	1300	1300	200		1 (c/C_P)
D_V_D	Periódica	100	100	30	C_P	3 (c/A_M)
E_D	Esporádica	2000	20	1		
R	Soft					

Tabela 5.1 - Definição das tarefas, versão inicial

O tempo de execução de alguns elementos do sistema operacional são computados juntos com o tempo de execução das tarefas. Por exemplo, o tempo de computação das chamadas de sistema e o tempo necessário para carregar o contexto da tarefa já estão considerados nos tempos de execução das tarefas.

O temporizador em hardware (*"timer"*) do sistema gera interrupções periodicamente. Estas interrupções são usadas para marcar a passagem do tempo e, entre outras coisas, liberar as tarefas sempre no início do respectivo período. Entretanto, a execução periódica do tratador de interrupções do *"timer"* representa uma interferência sobre as demais tarefas. Logo, o tratador de interrupções do *"timer"* entra no cálculo de escalonabilidade como uma tarefa periódica com período igual ao do *"timer"* e tempo de execução igual ao tempo de execução deste tratador de interrupções. São supostos os valores P_{timer}=10ms e C_{timer}=0.1ms para a tarefa *"timer"* que representa este tratador.

Também é necessário ampliar o modelo de tarefas inicial no sentido de incluir outros efeitos causados pelo suporte de execução. É suposto que a implementação emprega um sistema operacional de tempo real cujo *"kernel"* permanece quase a totalidade do tempo com as interrupções habilitadas. Entretanto, algumas seções críticas do *"kernel"* desabilitam as interrupções por breves instantes. A maior seção de código

do *"kernel"* que executa com interrupções desabilitadas o faz durante B_k = 0.1 ms.

Caso uma interrupção do *"timer"* aconteça enquanto o *"kernel"* está com interrupções desabilitadas, ela somente será tratada com um atraso (latência) de B_k. Suponha que esta interrupção do *"timer"* sinalize o início do período de uma tarefa. Neste caso, a liberação da tarefa somente vai acontecer B_k unidades de tempo após o início de seu período. Este atraso caracteriza claramente um *"release jitter"*. Todas as tarefas que não possuem predecessores sofrem um *"release jitter"* com duração máxima de B_k unidades de tempo, inclusive o próprio tratador do *"timer"*.

Outrossim, as tarefas que possuem predecessores não são liberadas por passagem de tempo mas sim pela sinalização da conclusão de sua tarefa predecessora. Desta forma, podemos considerar que a tarefa em questão possui um *"release jitter"* igual ao tempo máximo de resposta da sua tarefa predecessora. Observe que a tarefa predecessora é ativada pela passagem do tempo, e inclui no seu tempo máximo de resposta a latência de interrupção do sistema. A tarefa sucessora herda o *"release jitter"* da tarefa predecessora como parte do seu tempo máximo de resposta.

No caso do bloqueio por exclusão mútua, é importante observar que apenas ocorre bloqueio quando a tarefa de prioridade mais alta deve esperar a tarefa de prioridade mais baixa. Quando a tarefa de prioridade mais baixa tem que esperar, não ocorre bloqueio mas uma simples interferência. Assim, cada exclusão mútua gera apenas uma situação de bloqueio e não duas, como aparece na tabela 5.1. Por exemplo, a tabela 5.1 mostra bloqueio entre entre as tarefas R_R e C_P, e vice-versa. Na verdade somente existirá bloqueio da tarefa mais prioritária pela menos prioritária.

A tarefa esporádica E_D é assumida como periódica a nível de teste de escalonabilidade, como permitido pelas equações [9] e [10] do capítulo 2. Para executar a tarefa R podemos a princípio empregar qualquer tipo de servidor (PS ou DS, por exemplo). Mas a nível de teste ambos os servidores podem ser tratados como uma tarefa periódica. Se assumirmos que não serão pedidos dois relatórios em um intervalo de 10 segundos, então podemos tratar a tarefa R também como esporádica.

As tarefas E_D e R são ativadas por sinais de rádio e compartilham a mesma rotina que trata interrupções deste tipo. O tempo de execução deste tratador de interrupções é de B_m = 0.1 ms. Nesta aplicação o tratador de interrupções deste tipo tem o seu código acrescido ao da tarefa aperiódica esporádica E_D, pois é a tarefa mais prioritária que as demais da aplicação. Desta forma, ele aparece como interferência para as demais tarefas, com exceção da tarefa que representa o *"timer"*. Este tratador não interfere com o *"timer"* porque o vetor de interrupção é menos prioritário que o do *"timer"*. Entretanto, quando este tratador executa em função da tarefa R, ele aparece como uma situação de bloqueio para a tarefa E_D.

Será usada a política Deadline Monotônico (DM) neste sistema, pois algumas tarefas apresentam *"deadlines"* relativos menores que os seus respectivos períodos. Nas atividades a atribuição de prioridades obedece a orientação das relações de precedência,

isto é, tarefas predecessores possuem prioridade mais alta. A tabela 5.2 resume a definição das tarefas, após terem sido considerados todos os fatores discutidos acima. As tarefas aparecem na tabela 5.2 em ordem crescente de deadline, o que significa que as tarefas com prioridade mais alta aparecem antes na tabela.

Tarefa	Tipo	P (ms)	D (ms)	C (ms)	J (ms)	Predecessor	Bloqueio (ms)
timer	periódica	10	10	0,1	B_k		
C_P	periódica	100	100	20	B_k		1 (B_{R_R}))
L_I	periódica	500	500	20	B_k		
A_M	periódica	500	500	100	R_{L_I}	L_I	
R_R	periódica	1300	1300	200	B_k		
D_V_D	periódica	100	100	30	R_{C_P}	C_P	3 (B_{A_M})
E_D	esporádica	2000	20	1	B_k		B_m
R	servidora	10000	80	5	B_k		

Tabela 5.2: Definição das Tarefas, Versão Final

5.1.4 Teste de Escalonabilidade

Para verificar a escalonabilidade do conjunto usaremos o teste baseado em tempos de resposta apresentado em [TBW94], onde tarefas experimentam também atrasos em suas liberações. Este teste é constituído pelas equações [9] e [10] do capítulo 2, as quais são reproduzidas abaixo, e pela verificação da condição:

$\forall i : 1 \leq i \leq n, R_i \leq D_i$.

$$W_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i + J_j}{P_j} \right\rceil \cdot C_j \qquad [9]$$

$$R_i = W_i + J_i . \qquad [10]$$

Vamos a seguir calcular o tempo máximo de resposta de cada uma das tarefas da aplicação, como descritas no modelo de tarefas final. É importante destacar que o modelo de tarefas final inclui não somente as tarefas da aplicação, mas também tarefas que representam os custos associados com o suporte de execução.

Cálculo do tempo de resposta do *"timer"*

$$W_{timer} = C_{timer} = 0{,}1$$

$$R_{tiner} = W_{timer} + B_k = 0{,}2\,ms$$

Cálculo do tempo de resposta da tarefa E_D

$$W_{E\_D} = C_{E\_D} + B_M + \left\lceil \frac{W_{E\_D} + B_k}{P_{timer}} \right\rceil \times C_{timer} = 1{,}2$$

$$R_{E\_D} = W_{E\_D} + B_k = 1{,}3ms$$

Cálculo do tempo de resposta da tarefa R

$$W_R = C_R + \left\lceil \frac{W_R + B_k}{P_{timer}} \right\rceil \times C_{timer} + \left\lceil \frac{W_R + B_k}{P_{E\_D}} \right\rceil \times C_{E\_D} = 6{,}1$$

$$R_R = W_R + B_k = 6{,}2\,ms$$

Cálculo do tempo de resposta da tarefa C_P

$$W_{C\_P} = C_{C\_P} + B_{R\_R} + \left\lceil \frac{W_{C\_P} + B_k}{P_{timer}} \right\rceil \times C_{timer} + \left\lceil \frac{W_{C\_P} + B_k}{P_{E\_D}} \right\rceil \times C_{E\_D} +$$
$$+ \left\lceil \frac{W_{C\_P} + B_k}{P_R} \right\rceil \times C_R = 27{,}3$$

$$R_{C\_P} = W_{C\_P} + B_k = 27{,}4ms$$

Cálculo do tempo de resposta da tarefa D_V_D

$$W_{D\_V\_D} = C_{D\_V\_D} + B_{A\_M} + \left\lceil \frac{W_{D\_V\_D} + B_k}{P_{timer}} \right\rceil \times C_{timer} + \left\lceil \frac{W_{D\_V\_D} + B_k}{P_{E\_D}} \right\rceil \times C_{E\_D} + \\ + \left\lceil \frac{W_{D\_V\_D} + B_k}{P_R} \right\rceil \times C_R = 39{,}6$$

$$R_{D\_V\_D} = W_{D\_V\_D} + R_{C\_P} = 67\,ms$$

Cálculo do tempo de resposta da tarefa L_I

$$W_{L\_I} = C_{L\_I} + \left\lceil \frac{W_{L\_I} + B_k}{P_{timer}} \right\rceil \times C_{timer} + \left\lceil \frac{W_{L\_I} + B_k}{P_{E\_D}} \right\rceil \times C_{E\_D} + \left\lceil \frac{W_{L\_I} + B_k}{P_R} \right\rceil \times C_R + \\ + \left\lceil \frac{W_{L\_I} + B_k}{P_{C\_P}} \right\rceil \times C_{C\_P} + \left\lceil \frac{W_{L\_I} + R_{C\_P}}{P_{D\_V\_D}} \right\rceil \times C_{\mathrm{D\_V\_D}} = 123{,}3$$

$$R_{L\_I} = W_{L\_I} + B_k = 127{,}4\,ms$$

Cálculo do tempo de resposta da tarefa A_M

$$W_{A\_M} = C_{A\_M} + \left\lceil \frac{W_{A\_M} + B_k}{P_{timer}} \right\rceil \times C_{timer} + \left\lceil \frac{W_{A\_M} + B_k}{P_{E\_D}} \right\rceil \times C_{E\_D} + \left\lceil \frac{W_{A\_M} + B_k}{P_R} \right\rceil \times C_R + \\ + \left\lceil \frac{W_{A\_M} + B_k}{P_{C\_P}} \right\rceil \times C_{C\_P} + \left\lceil \frac{W_{A\_M} + R_{C\_P}}{P_{D\_V\_D}} \right\rceil \times C_{\mathrm{D\_V\_D}} = 258{,}6$$

$$R_{A\_M} = W_{A\_M} + R_{L\_I} = 386{,}0\,ms$$

Cálculo do tempo de resposta da tarefa R_R

$$W_{R\_R} = C_{R\_R} + \left\lceil \frac{W_{R\_R} + B_k}{P_{timer}} \right\rceil \times C_{timer} + \left\lceil \frac{W_{R\_R} + B_k}{P_{E\_D}} \right\rceil \times C_{E\_D} + \left\lceil \frac{W_{R\_R} + B_k}{P_R} \right\rceil \times C_R +$$
$$+ \left\lceil \frac{W_{R\_R} + B_k}{P_{C\_P}} \right\rceil \times C_{C\_P} + \left\lceil \frac{W_{R\_R} + R_{C\_P}}{P_{D\_V\_D}} \right\rceil \times C_{\mathrm{D\_V\_D}} + \left\lceil \frac{W_{R\_R} + B_k}{P_{L\_I}} \right\rceil \times C_{L\_I} +$$
$$+ \left\lceil \frac{W_{R\_R} + R_{L\_I}}{P_{A\_M}} \right\rceil \times C_{A\_M} = 1228{,}3$$

$$R_{R\_R} = W_{R\_R} + B_k = 1228{,}4\,ms$$

Como todas as tarefas apresentam tempo máximo de resposta menor do que o respectivo deadline (por exemplo, a tarefa R_R tem um tempo máximo de resposta de 1228.4 milisegundos, enquanto o seu deadline é de 1300ms), conclui-se que o sistema é escalonável.

Na próxima seção, serão discutidos aspectos da implementação desta aplicação e entre outras coisas será, a título de ilustração, definido como suporte deste projeto o RT-Linux. Como na abordagem assíncrona é necessário que se considere aspectos de implementação nas análises, todos os tempos referentes ao *"kernel"* devem sempre ser considerados tomando como base o suporte escolhido. Os tempos referentes aos tratadores e bloqueios do *"kernel"* (B_m, C_{timer} e B_k) foram super estimados quando assumidos nas nossas análises como sendo da ordem de 0.1ms. As medidas de latência de interrupção apresentadas em [YoB99] para a versão 2 do RT-Linux são da ordem de microsegundos. Este pessimismo exagerado dá uma boa margem de segurança que, uma vez o conjunto de tarefas passe pelos testes de escalonabilidade, estas tarefas quando implementadas terão suas restrições temporais respeitadas em tempo de execução.

5.1.5 Programação Usando RT-Linux

Uma aplicação de tempo real é tipicamente um programa concorrente. Logo, todas as técnicas de programação e métodos de depuração usados na programação concorrente podem ser usados aqui. Em particular, o ambiente de programação escolhido define como a solução será expressa em termos sintáticos. Ambiente de programação neste contexto significa a linguagem de programação e o sistema operacional escolhido. O tema "linguagens de programação" não será abordado neste livro, embora seja um tópico importante (um levantamento neste sentido pode ser

encontrado em [Coo96]). Apesar de diversas propostas a nível acadêmico, aplicações de tempo real são ainda programadas principalmente com C e C++. Linguagens de programação especialmente projetadas para tempo real procuram embutir na própria sintaxe da linguagem as construções relacionadas com tempo e concorrência, mas isto não acontece com C e C++. Estas são linguagens a princípio para programação seqüencial e qualquer capacidade além desta é obtida através de chamadas de rotinas de uma biblioteca ou do sistema operacional.

Com o propósito de ilustrar o tipo de serviço encontrado na prática, assim como a sintaxe típica, esta seção descreve parcialmente a API (*"Application Programming Interface"*) do Real-Time Linux versão 2, como descrita em [YoB99] e disponível, assim como o próprio sistema operacional, na página http://www.rtlinux.org. O RT-Linux está em rápida evolução e, provavelmente, quando este livro for publicado, novas versões já estarão disponíveis. A versão 2 do RT-Linux, implementada para processadores x86 genéricos, apresenta uma latência de interrupção máxima de 15 microsegundos. Uma tarefa periódica inicia sua execução com um desvio máximo de 25 microsegundos do instante planejado (se não existir outra tarefa liberada com prioridade maior). São valores que permitem o emprego do RT-Linux em uma ampla variedade de aplicações.

O conjunto de rotinas do RT-Linux é suficiente para implementar uma aplicação de tempo real composta por várias tarefas. Este sistema operacional foi projetado a partir da premissa que tarefas de tempo real executam sobre o *"microkernel"* com funcionalidade mínima e excelente comportamento temporal, ao passo que tarefas sem restrição temporal executam como tarefas Linux convencionais. Desta forma, qualquer acesso a disco, tela ou rede local deve ser feito por tarefas convencionais. A comunicação entre estes dois tipos de tarefas pode ser feita através do RT-FIFO ou através de memória compartilhada.

Abaixo está o exemplo de uma tarefa periódica simples, usando a API do RT-Linux versão 1. Neste exemplo foram utilizadas as seguintes chamadas de sistema:

int rt_get_time (void);

Retorna a hora em *"ticks"*.

int rt_task_init (RT_TASK *task, void (*fn)(int data), int data,
int stack_size, int priority);

Inicializa mas não escalona uma tarefa.

int rt_task_make_periodic (RT_TASK *task, RTIME start_time,
RTIME period);

Transforma a *"thread"* em periódica.

int rt_task_wait (void);

Libera o processador até a próxima ativação da tarefa.

```
#define     STACK_SIZE  3000
RT_TASK     my_task;

/* Código da tarefa de tempo real, um laço infinito */
void codigo_da_tarefa( unsigned int x) {
      /* Variáveis locais desta tarefa */
      . . .
      while ( 1 ) {
            /* Faz o que esta tarefa tem que fazer */
            . . .
            /* Espera até o próximo período */
            rt_task_wait();
            }
}

/* Módulo de inicialização da aplicação */
int init_module( void ){
      RTIME now = rt_get_time();

      /* Inicializa a tarefa */
      rtl_task_init( &my_task, codigo_da_tarefa, arg, STACK_SIZE, 1 );

      /* Executa a cada 50 ms, aproximadamente 450 na unidade usada */
      rtl_task_make_periodic(&mytask, now, 450);
      return 0;
}
```

Um exemplo completo usando RT-Linux pode ser encontrado em [Wur99], onde uma aplicação de controle é descrita. Uma interface gráfica de usuário usando X-Windows comunica-se com tarefas de tempo real através de RT-FIFO, as quais controlam uma variável física através de um algoritmo de controle simples.

5.2 Aplicação com Abordagem Síncrona

O objetivo deste exemplo é de ilustrar uma metodologia e um estilo de programação para a abordagem síncrona. A aplicação escolhida é simples o suficiente para o leitor não precisar de conhecimentos adicionais e poder focalizar apenas os aspectos associados a sua programação. A aplicação consiste de um sistema de controle de processo, inspirado no exemplo apresentado em [AnP93] e cuja descrição segue. O sistema a controlar consiste de um reservatório de água do qual pretende-se controlar o nível para que ele fique sempre entre um mínimo e um máximo. Uma bomba quando ligada tende a esvaziar o reservatório e uma válvula permite enche-lo quando aberta. A informação de nível é fornecida por dois sensores indicando os limites inferior e superior para a água contida no reservatório (respectivamente nível mínimo e nível máximo). A figura 5.2 mostra um diagrama esquemático da aplicação de regulação de nível de um reservatório.

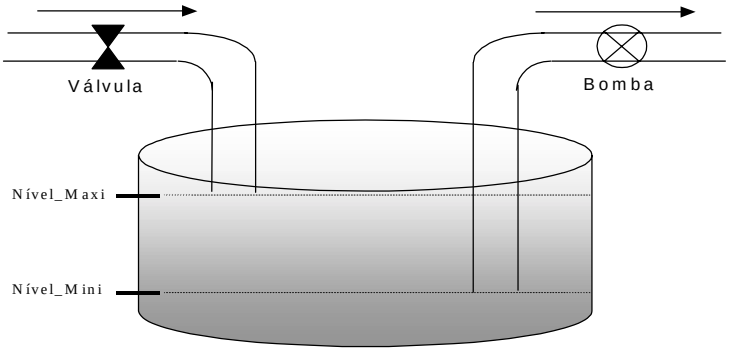

Figura 5.2: Esquema da Regulação de Nível de um Reservatório

O sistema de controle será estudado e construído num primeiro tempo para o comportamento normal do sistema a controlar e a seguir para uma situação de falha neste. Inicialmente, é tratado o caso do funcionamento normal do sistema a controlar. O problema é decomposto em duas partes, uma correspondente a atividade da válvula para regular o nível do reservatório e outra a atividade de consumo de água pela bomba. Essas duas atividades são independentes e ocorrem em paralelo. A partir desta visão do problema, é possível construir o programa do sistema de controle do reservatório como sendo o resultado da composição de dois módulos atuando em paralelo, um responsável pela regulação de nível a partir da ação da válvula (**REGNIVEL**) e outro pelo consumo de água pela bomba (**CONSUMO**). A arquitetura do sistema de controle é apresentada na figura 5.3.

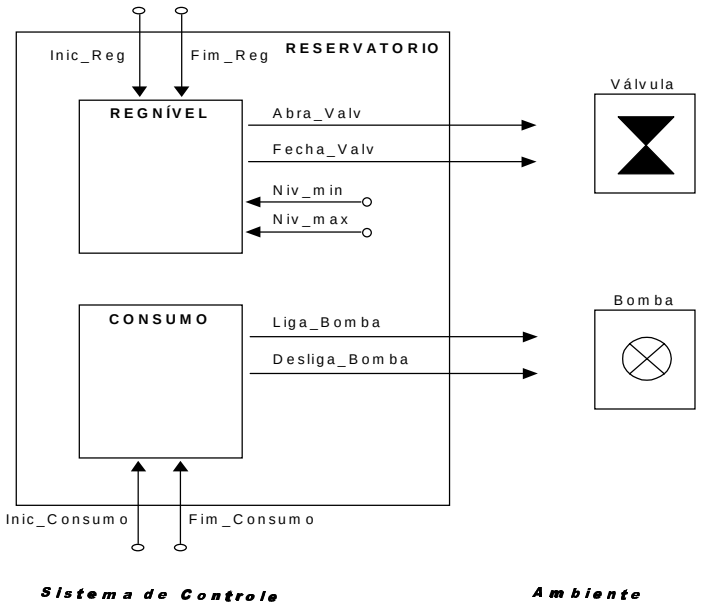

Figura 5.3: Arquitetura do Sistema de Controle

O módulo **REGNIVEL** visa manter o nível do reservatório entre o mínimo e o máximo indicado pelos sensores de nível (sinais **Niv_min** e **Niv_max),** abrindo e fechando a válvula de regulação do reservatório a partir da emissão dos sinais **Abra_Valv** e **Fecha_Valv**. O mecanismo de regulação é ativado por um comando externo de inicio (sinal **Inic_Reg)** e atua até ocorrer o comando de fim (sinal **Fim_Reg)**. Para efeito de simplificação do problema, é assumido que no inicio, o reservatório se encontra com um nível intermediário entre o mínimo e o máximo. O código em Esterel do módulo é apresentado a seguir:

```
module REGNIVEL :
input Inic_Reg, Fim_Reg, Niv_min, Niv_max;
output Abra_Valv, Fecha_Valv;

await Inic_Reg;
emit Fecha_Valv;
abort
        loop
                await Niv_min;
                emit Abra_Valv;
                await Niv_max;
                emit Fecha_Valv;
        endloop
when Fim_Reg
endmodule
```

Deve se notar que a programação do módulo **REGNIVEL** segue o estilo de programação Esterel descrito no capítulo 4. Após a inicialização do processo de regulação a partir do sinal **Inic_Reg**, o corpo do programa descrevendo a regulação de nível pela válvula está embutido numa construção do tipo preempção forte que termina com a chegada do sinal **Fim_Reg**.

O módulo **CONSUMO** gera os comandos de liga e desliga da bomba de água (sinais **Liga_Bomba, Desliga_Bomba**), a partir de comandos externos de inicio e fim de consumo (sinais **Inic_Consumo**, **Fim_Consumo**). O código deste módulo em Esterel é:

```
module CONSUMO :
input Inic_Consumo, Fim_Consumo;
output Liga_Bomba, Desliga_Bomba;

loop
        await Inic_Consumo;
        emit Liga_Bomba;
        await Fim_Consumo;
        emit Desliga_Bomba
endloop
endmodule
```

O programa em Esterel do sistema de controle do Reservatório em situação de funcionamento normal do sistema a controlar é o módulo **RESERVATORIO**, resultado da composição paralela dos dois módulos anteriores:

```
module RESERVATORIO :
input Inic_Reg, Fim_Reg, Inic_Consumo, Fim_Consumo;
output Abra_Valv, Fecha_Valv, Liga_Bomba, Desliga_Bomba;

signal Niv_min, Niv_max in
        run CONSUMO
||
        run REGNIVEL
end signal
end module
```

No caso de considerar também as situações anormais do sistema a controlar, o sistema necessita além do módulo de controle, um módulo de detecção de falha e um de tratamento desta. Uma situação anormal clássica neste exemplo do reservatório corresponde ao funcionamento da bomba "a vazio"; é necessário detectar então quando o nível do reservatório é muito baixo para poder parar o funcionamento da bomba, evitando danifica-la. Para descrever o comportamento do sistema, torna-se necessário introduzir um módulo encarregado de detectar falha (no caso o nível de água insuficiente) e um módulo de recuperação da falha. Esses módulos são respectivamente os módulos **MONITORA_CONSUMO** e **REPARO**.

Para ter a garantia que existe realmente uma falha no sistema, o mecanismo de detecção do módulo **MONITORA_CONSUMO** atuará somente se o sinal **Niv_min** estiver presente em três ocorrências seguidas de um sinal de relógio **Rel**, confirmando desta forma a existência da falha. A ocorrência desta falha gera uma exceção que interrompe imediatamente o bloco do mecanismo de detecção e passa o controle para o bloco de tratamento de exceção. A construção Esterel "**trap T ... exit T ... handle T ...**" é perfeitamente adequada para implementar o mecanismo de levantamento de exceção ("**trap T ... exit T**") e o tratamento desta ("**handle T**"); o módulo de detecção de falha **MONITORA_CONSUMO** esta construído a partir dela. O corpo da construção **trap** contém o mecanismo de detecção de falha adotado que quando acionado, desvia para o tratador de exceção desta **handle**. O módulo **MONITORA_CONSUMO** tem o código mostrado na próxima página.

O tratamento da exceção levantada no módulo **MONITORA_CONSUMO** é realizado através do módulo **REPARO** que se encarrega do reparo da situação anormal correspondente ao nível insuficiente do reservatório. A especificação do sistema define que este reparo deve ser realizado num tempo inferior a 10 minutos, senão um alarme para um nível mais alto de supervisão deverá ser enviado para assinalar a impossibilidade de reparo.

```
module MONITORA_CONSUMO :
input Rel, Niv_min, Consumo_em_Curso

var contador_tempo: integer in

contador_tempo := 0;
loop
        trap MONITOR in
                loop
                        present Niv_min then
                                contador_tempo := contador_tempo + 1;
                                present Consumo_em_Curso then
                                        if contador_tempo >= 3 then
                                                exit MONITOR
                                        end if
                                end present
                        else
                                contador_tempo := 0
                        end present
                each Rel
        handle MONITOR do
        % tratamento da situação detectada de insuficiência do nível de água
        % pela execução do módulo REPARO
        end trap
end loop
end var
end module
```

O reparo é realizado por uma tarefa assíncrona **Reparo_Nivel_Insuficiente ( ) ( )** do módulo **REPARO** que iniciará logo após a exceção **MONITOR** do módulo **MONITORA_CONSUMO** ter sido levantada. A execução desta tarefa assíncrona externa (não descrita aqui) é realizada utilizando a primitiva **exec** de Esterel. No mesmo tempo que inicia esta tarefa**,** o módulo **REPARO** envia o sinal **Bomba_em_Reparo** a todos os outros módulos, segundo o mecanismo de difusão instantânea de Esterel, para impedi-los de atuar enquanto a situação de reparo estiver mantida. A finalização da tarefa assíncrona de reparo **Reparo_Nivel_Insuficiente ( ) ( )** pode ser feita de forma normal se a duração de reparo for inferior a 10 minutos, emitindo um sinal de **Reparo_Bomba_Concluido** que permitirá abortar a execução concorrente, autorizando novamente a atuação dos outros módulos. Se o tempo de reparo ultrapassar o tempo fixado de 10 minutos, um *"time-out"* é detectado, ativando a emissão de um sinal de alarme **Alarme_Bomba** no nível superior de supervisão e aguardando que este lhe permita retomar a execução no futuro através de um sinal **Bomba-Autorizada** vindo do nível de supervisão. O código do módulo **REPARO** é apresentado a seguir:

```
module REPARO :
input MINUTE, Bomba-Autorizada;
output Bomba_em_Reparo, Alarme_Bomba, Reparo_Bomba_Concluido;
task Reparo_Nivel_Insuficiente ( ) ( );

        [
                abort
                        exec Reparo_Nivel_Insuficiente ( ) ( )
                when 10 MINUTE do
                        emit Alarme_Bomba;
                        await Bomba-Autorizada
                end abort;
                emit Reparo_Bomba_Concluido
        ]
||
        abort
                sustain Bomba_em_Reparo
        when Reparo_Bomba_Concluido
endmodule
```

Deve se notar que o módulo **REPARO** é composto de dois blocos em paralelo, um deles responsável pela execução do reparo da bomba e o outro pela informação aos outros módulos (via difusão instantânea) da situação de reparo desta, enquanto durar. A informação de conclusão do reparo no primeiro bloco (sinal **Reparo_Bomba_Concluido**) é imediatamente difundida ao segundo que deixa de emitir para os outros módulos o sinal **Bomba_em_Reparo**.

Para levar em conta, no programa em Esterel do sistema de controle do Reservatório, o comportamento faltoso e sua recuperação descritos anteriormente é necessário substituir no módulo **RESERVATORIO** o módulo **CONSUMO** por um módulo **CONSUMO_SEGURO**. Neste módulo, a indicação de ativação e desativação do sistema é feita respectivamente a partir dos sinais **Inic** e **Fim**. No módulo **CONSUMO_SEGURO**, após a ativação do sistema (sinal **Inic**), é emitido o sinal de inicio da regulação (sinal **Inic_Reg)** que será interrompida quando o sinal de desativação do sistema ocorrer (sinal **Fim**). Entre essas duas ocorrências (ativação e desativação), o módulo **MONITORA_CONSUMO** atuará em paralelo com o bloco que descreve o consumo de água e emite comandos para ligar e desligar a bomba. O código do módulo **CONSUMO_SEGURO** é apresentado a seguir:

```
module CONSUMO_SEGURO :
input Inic, Fim, Inic_Consumo, Fim_Consumo, Rel, Niv_min, MINUTE, Bomba-Autorizada;
output Liga_Bomba, Desliga_Bomba, Alarme_Bomba, Inic_Reg, Fim_Reg;
task Reparo_Nivel_Insuficiente ( ) ( );

signal Bomba_em_Reparo, Consumo_em_Curso, Reparo_Bomba_Concluido in
        await Inic;
        emit Inic_Reg;
        abort
                loop
                        await Inic_Consumo;
                        emit Liga_Bomba;
                        abort
                                sustain Consumo_em_Curso
                        when Fim_Consumo;
                        emit Desliga_Bomba
                end loop
        ||
                run MONITORA_CONSUMO
        when Fim
end signal
end module
```

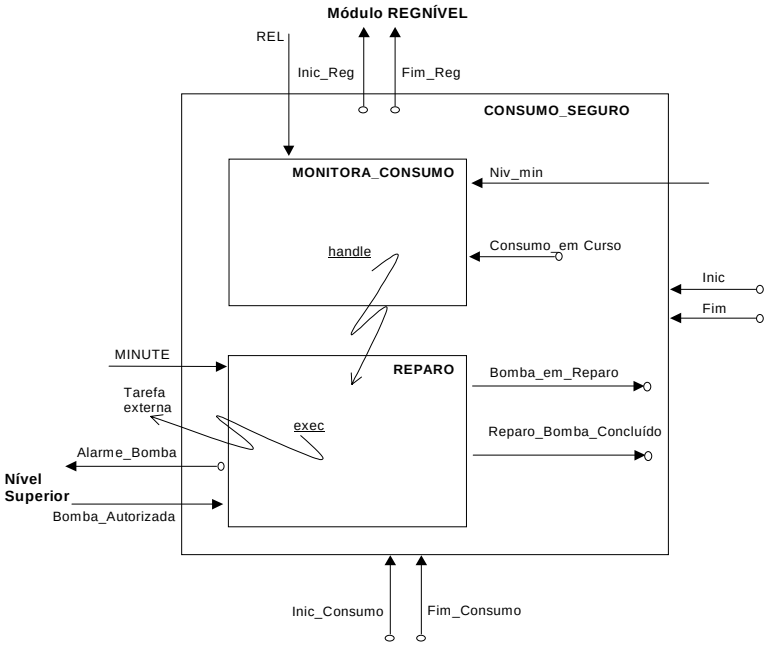

Figura 5.4: Representação do Módulo **CONSUMO_SEGURO**

A figura 5.4 apresenta o módulo **CONSUMO_SEGURO** com seus módulos internos **MONITORA_CONSUMO** e **REPARO** e suas interfaces com o módulo **REGNIVEL**, com o nível superior de Supervisão e com o ambiente (ou sistema a controlar).

No exemplo anterior, foi apresentado um sistema simples no qual a modelagem do comportamento não apresentava muitas alternativas. Entretanto, dependendo da complexidade do problema, podem existir várias possibilidades de modelar o comportamento deste. A seguir, são apresentadas e discutidas algumas abordagens possíveis para a modelagem de sistemas mais complexos como por exemplo sistemas de comando de uma célula de manufatura composta de vários equipamentos (robôs, maquinas, esteiras, sensores, etc.). Exemplos de tais aplicações podem ser encontrados em [Cos 89], [Bud94], [CER92] e servem de base para a apresentação das diversas abordagens de modelagem:

Abordagem 1: Decomposição por componente físico
Esta abordagem consiste em associar um módulo Esterel a cada componente físico do problema (p.ex. um módulo para cada robô) e em aproveitar o conceito de paralelismo da linguagem Esterel. Cada módulo é constituído por um conjunto de atividades geralmente realizadas em seqüência (p.ex. o comportamento de um robô pode ser visto como a realização de uma atividade somente após a conclusão de outra). Todos estes módulos são colocados em paralelo e a troca de sinais entre eles permitem que eles cooperam entre si.

Abordagem 2: Decomposição funcional
A abordagem anterior esta baseada numa visão totalmente paralela da aplicação, com alguns pontos de sincronização. Entretanto, esta visão do problema não leva em conta na modelagem a existência de tarefas da aplicação que se executam seqüencialmente. A segunda abordagem consiste em programar a aplicação a partir de uma decomposição funcional desta. A idéia básica desta abordagem consiste em considerar que algumas tarefas só podem ser realizadas após o término de outras; a modelagem da aplicação e sua conseqüente programação leva em conta este fato e só permite o inicio da execução das varias tarefas quando necessário, evitando a espera de eventos que não tem nenhuma probabilidade de ocorrer.

Abordagem 3: Decomposição por recursos compartilhados
Esta abordagem segue a decomposição funcional como na abordagem anterior, entretanto ela leva em conta a existência de recursos comuns e decompõe a aplicação em função destes. Cada uma das funcionalidades que utilizam um mesmo recurso corresponde a um módulo; todos esses módulos são colocados em paralelo.

Essas três abordagens foram comparadas nos trabalhos citados anteriormente do ponto de vista da complexidade do autômato resultante (i.e. número de estados e de

transições). Os resultados obtidos mostram diferenças grandes (da ordem de 10 a 20 vezes) entre o número de estados e de transições dos autômatos resultantes da abordagem 1 e das abordagens 2 e 3, apesar de descrever as mesmas funcionalidades. A arquitetura adotada na abordagem 1 é a causa deste aumento no número de estados, pois por causa do paralelismo, os módulos esperam um grande número de sinais (eventos) de entrada; como parte deles são impossíveis de serem recebidos antes de outros terem sido emitidos, a abordagem 1 cria um grande número de estados nunca atingidos e então desnecessários. Por outro lado, as abordagens 2 e 3 implicam em uma implementação centralizada enquanto a abordagem 1 permite também adotar uma implementação distribuída, com programas separados para cada módulo-equipamento. Estas considerações devem ser levadas em conta quando da programação de sistemas complexos e fazem parte da metodologia a ser adotada para programar aplicações segundo a abordagem síncrona.

5.3 Considerações sobre as abordagens usadas

Os exemplos apresentados neste capítulo mostram que as abordagens assíncrona e síncrona apresentam características muito diferentes e consequentemente existem campos de aplicação para as quais cada uma delas é mais apropriada. Esta seção procura destacar estas diferenças, assim como indicar critérios utilizados para selecionar uma ou outra abordagem frente a uma dada aplicação. Essas abordagens, as quais visam fornecer soluções para o problema de tempo real, devem ser analisadas tanto do ponto de vista da especificação e verificação quanto da implementação.

A abordagem assíncrona é considerada como orientada à implementação e consequentemente descreve o sistema da forma a mais completa possível. Essa descrição leva em conta a ocorrência e a percepção de todos os eventos numa ordem arbitrária mas não simultânea; o comportamento observado corresponde a todas as combinações de ocorrência de eventos (*"interleaving")*. A abordagem síncrona é considerada como orientada ao comportamento da aplicação e a sua verificação. A premissa básica da abordagem síncrona é a simultaneidade dos eventos de entrada e saída, partindo da hipótese que no nível de abstração considerado, cálculos e comunicações não levam tempo. A abordagem síncrona se situa num nível de abstração dos aspectos de implementação tal que essa hipótese seja verdadeira.

Como conseqüência destes princípios e modelos diferentes nas duas abordagens, leva-se em conta na especificação e no projeto segundo a abordagem assíncrona, características do suporte de software e de hardware das aplicações, o que pode dificultar requisitos de portabilidade. A abordagem síncrona é mais distante das questões de implementação, o que lhe dá uma vantagem do ponto de vista da portabilidade.

A descrição completa do comportamento e a introdução de considerações de implementação na abordagem assíncrona torna complexa a análise das propriedades do

sistema por causa do grande número de estados gerados e do possível não determinismo introduzido. A abordagem síncrona, se situando num nível de abstração maior, facilita a especificação e a verificação das propriedades de sistemas de tempo real. Como resultado, os sistemas construídos a partir da abordagem síncrona são muito mais confiáveis do que aqueles construídos segundo a abordagem assíncrona. A abordagem síncrona, entretanto por se situar num nível de abstração maior do que a abordagem assíncrona pode encontrar dificuldades em algumas situações do ponto de vista da implementação da qual está mais afastada, em particular em implementações que necessitam ser distribuídas.

Na abordagem assíncrona, a concorrência e o tempo estão tratados de forma explícita, tornando fundamental o estudo da previsibilidade dos sistemas de tempo real e consequentemente a questão do escalonamento tempo real. As premissas da abordagem síncrona permitem resolver a concorrência sem o entrelaçamento de tarefas, não existindo a necessidade de escalonamento. Esta surge quando diferentes atividades requerem o uso do mesmo recurso (processador, qualquer periférico ou até mesmo as estruturas de dados), o qual não pode ser usado simultaneamente pelas várias atividades. Na abordagem síncrona, como todas as atividades são instantâneas, o recurso não é ocupado, e deixa de ser um problema. Além do mais, na abordagem síncrona, o tempo não aparece de maneira explícita.

Na abordagem síncrona, a única preocupação do projetista da aplicação é garantir a resposta correta aos eventos do ambiente, no sentido lógico. A correção no sentido temporal está automaticamente garantida pois, como a velocidade de processamento é considerada infinita, o tempo de resposta será sempre zero. A abordagem assíncrona parte da programação concorrente clássica e procura tratar da questão tempo, além das questões de sincronização entre tarefas. Para isto as políticas de escalonamento são tornadas explícitas (algo que não acontece na programação concorrente clássica) e os mecanismos de sincronização (semáforos por exemplo) são dotados de um comportamento mais apropriado para a análise de escalonabilidade. Interferências, precedências e bloqueios são considerados no momento de determinar se os *"deadlines"* da aplicação serão ou não cumpridos. Um sistema construído segundo a abordagem assíncrona não precisa ser mais rápido que o ambiente a sua volta. Ele apenas precisa, usando os recursos de forma inteligente, atender aos requisitos temporais (*"deadlines"*) expressos na especificação do sistema. Na abordagem síncrona esta análise de escalonabilidade usada na abordagem assíncrona torna-se trivial e desnecessária, pois os *"deadlines"* são muitas ordens de grandeza maiores do que o tempo que o processador leva para executar as tarefas da aplicação.

O campo de aplicação da abordagem síncrona se limita a situações de sistemas ou partes de sistemas nas quais a velocidade do ambiente pode ser considerada como menor que a do computador, atendendo as exigências da hipótese de sincronismo. Nestas situações, a abordagem síncrona se impõe pela sua simplicidade, facilidade de especificação e potencialidade em verificar o sistema especificado. Esta abordagem não necessita de um processo de tradução para passar da especificação validada ao programa a ser executado; ela implementa diretamente a especificação, o que se

configura numa grande vantagem por evitar a introdução de erros durante o processo de tradução. Uma aplicação construída com a abordagem assíncrona é mais complexa e mais difícil de verificar e consequentemente mais sujeita a falhas do que a mesma aplicação na abordagem síncrona. Entretanto, quando o computador não consegue ser muito mais rápido que o ambiente ou quando não é possível determinar com garantia a relação de tempo entre o ambiente e o computador (p.ex. é o caso da Internet), não existe escolha e a abordagem assíncrona deve ser utilizada. Geralmente nos sistemas complexos, as duas abordagens podem se tornar necessárias; partes mais reativas do sistema tais como interfaces homem-máquina, controladores, protocolos de comunicação e "drivers" de periféricos podem ser tratadas pela abordagem síncrona, enquanto as partes correspondendo aos cálculos, ao manuseio e tratamento de dados e à comportamentos não-deterministas tem que fazer uso da abordagem assíncrona.

5.4 Conclusão

Este capítulo apresentou duas aplicações de tempo real. A aplicação "veículo com navegação autônoma" foi implementada através da abordagem assíncrona, com análise de escalonabilidade e emprego de um sistema operacional de tempo real. A aplicação "sistema de controle" foi implementada através da abordagem síncrona, com o emprego da linguagem Esterel. Finalmente, a seção 5.3 procurou fazer uma comparação entre as duas abordagens.

Embora a limitação de espaço permita apenas uma descrição superficial destas aplicações, elas permitem que o leitor tenha contato com o método implícito na adoção de cada uma das abordagens. A partir destes exemplos, fica mais fácil identificar situações onde uma e outra podem ser melhor aplicadas.