Sistemas de Tempo Real

Nuno Guimarães e Nuno Nunes

26 de Dezembro de 2001

Conteúdo

Capítulo 1

O ambiente de tempo real

1.1 Quando é que um sistema computurizado é de tempo real?

Um sistema computurizado de tempo real é um sistema em que o correcto desempenho é avaliado não só pela resposta adequada a um dado estímulo mas também se essa resposta é dada em tempo útil, isto é, quando ela é necessária.
Ideia a reter: *Cumprir Prazos!*
Um sistema computorizado de tempo real é parte integrante de um sistema mais amplo, um Sistema de Tempo Real, composto da seguinte forma:

- Interface humana (man-machine interface)
- Sistema computorizado de tempo real
- Interface de instrumentação (sensores e actuadores)
- Objecto a controlar

Sistema de tempo real distribuído: são nós interligados por uma rede de comunicação de tempo real.

Deadline: tempo máximo para o sistema realizar uma determinada acção, em função de um determinado estímulo a partir do qual se contabiliza a deadline.
Uma deadline é considerada firme se o resulatado de ser ultrapassado é a catástrofe (exemplo: semáforo numa passagem de nível). Tais sistemas, em que exista pelo menos uma deadline firme, classificam-se como Hard

Real Time Systems.

1.2 Requisitos funcionais

Dividem-se em 3 grupos:

1. Requisitos de colheita de informação
2. Requisitos de controlo digital
3. Requisitos de interacção homem-máquina (man-machine)

1.2.1 Requisitos de colheita de informação

Imagem de tempo real: é uma amostra de uma entidade de tempo real. O foco e dado à ideia de que a imagem tem um determinado período de validade, findo o qual já não corresponde ao estado da variável ou, pelo menos, não o podemos garantir.

Intervalo de segurança ou fiabilidade: intervalo em que a amostra (imagem da entidade de tempo real) é válida.

About signal conditioning... Ideia essencial: quando um sensor fornece uma amostra é necessário, primeiro, validá-la por um processo qualquer (por exemplo, por comparação com amostras anteriores) de modo a detectar falhas do sensor.

About alarm monitoring... Porque numa situação de alarme muitas variáveis vão ter comportamento estranho, então torna-se necessário monitorizar de forma a detectar a anomalia que despoletou a situação de alarme.
Em muitas situações torna-se mesmo essencial prever o comportamento do sistema em situações anómalas.

Time Triggered: observação despoletada periodicamente por um relógio de tempo real.

Event Triggered: observação despoletada por eventos (alterações do estado).

1.2.2 Requisitos de controlo digital

Quando o controlo é exercido sem correr aos sistemas de controlo convencionais temos controlo directo de computador sobre os actuadores. Realiza uma sequência cíclica (amostragem -> algorítmo de controlo -> actuação) capaz de satisfazer os objectivos de controlo e compensar as perturbações do sistema.

1.2.3 Interacção homem-máquina

É importante na medida em que o operador interfere com o sistema baseado na informação apresentada por esta interface. Uma apresentação mal estruturada pode conduzir a interpretações erradas sobre o estado do processo.

1.3 Requisitos temporais

1.3.1 O que são

São os requisitos impostos pela teoria dos sistemas realimentados.

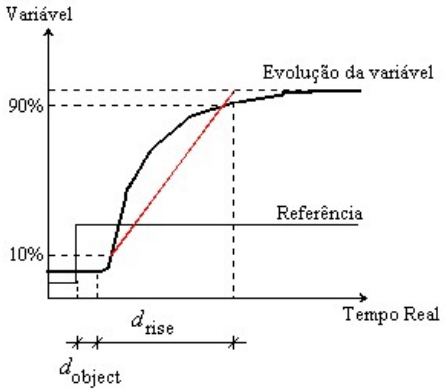

Figura 1.1: Atraso e tempo de subida de uma resposta ao degrau

Regra: d_{sample}, ou seja período de amostragem, deve ser menor que 1/10 do tempo de subida:

$$d_{sample} < \frac{d_{rise}}{10}$$

$d_{computer}$- atraso em relação ao instante de amostragem. Convém que

$$d_{computer} < d_{sample}$$

A diferença entre valor máximo e mínimo de dcomputer chama-se *jitter*. Este *jitter* é um parâmetro de qualidade de controlo.
Deadtime: é a soma de $d_{object} + d_{computer}$ representa o intervalo entre um instante de observação e o início da reacção do objecto à acção do computador baseada nesta observação.

1.3.2 Requisitos mínimos ao valor do *jitter*

O *jitter* introduz um grau de incerteza acerca do instante em que se deu a observação. Deste modo representa um erro adicional ao valor da medida; é pois necessário que o *jitter* seja sempre uma pequena fracção do atraso.

1.3.3 Requisitos mínimos do tempo de detecção de erro

Numa aplicação de tempo real *hard* é necessário detectar erros no mais curto espaço de tempo e com grande probabilidade. O tempo necessário para detectar o erro deve ser da mesma ordem de grandeza do período de amostragem do sistema de controlo mais rápido, de modo a que seja temporalmente possível exercer uma acção correctiva sobre o sistema.

1.4 Desempenho

São os requisitos dos quais depende a qualidade do serviço prestado pelo sistema computurizado.

1.4.1 Confiança (reliability)

É a probabilidade do sistema fornecer o serviço para que foi especificado até ao instante t.

$$R(t) = exp(-L(to)), \qquad L = \text{falhas por unidade de tempo}$$

1.4.2 Segurança (safety)

Safety é o mesmo que *reliability* para modos de falha críticos (malígnos). É muito importante para sistemas onde o custo de uma falha é enorme (exemplo: falha do sistema de travagem num automóvel).Uma vez mais mede se em número de falhas por unidade de tempo.

Maintainability - Característica do sistema que mede o tempo necessário para reparar uma falha benígna.

Availability - Característica que mede o tempo durante o qual o sistema está "apto"a fornecer serviço. É calculada pela expressão:

$$\begin{aligned} Availability(A) &= \frac{MTTF}{MTTF + MTTR} \\ MTTF &= \text{mean time to fail} \\ MTTR &= \text{mean time to repair} \end{aligned}$$

Security - Determina a segurança do sistema contra invasões não autorizadas. É um restrição ao acesso (por password, por exemplo).

1.5 Classificação de sistemas de tempo real

1.5.1 Sistemas hard RT versus soft RT

Tempo de resposta: é suposto que se uma deadline é ultrapassada num sistema soft real time não ocorrerá nenhuma catástrofe. Logo, existe uma certa margem de manobra.

Desempenho em situação anómala: um sistema hard real time deve continuar a cumprir deadlines. Nestes sistemas estas sistuações devem ser antecipadas e previstas durante o projecto.

Segurança: O sistema deve ser capaz de detectar sozinho os erros que surgem.

Tamanho dos ficheiros de dados: ficheiros de pequena dimensão que induzam tempos de processamento curtos.

1.5.2 Fail-safe vs fail-operation

Fail-safe: o sistema, na ocorrência de erros, transita para um estado que o projectista considera "seguro".

Fail-operational: o sistema deve permitir que durante situações de falha ainda consiga operar. É o caso de sistemas onde não se pode à partida definir um estado seguro.

1.5.3 Granteed response vs Best effort

Granteed response: prevêm-se as situações de falha e cobrem-se essas falhas no projecto. A hipótese do sistema não ser capaz de lidar com situações de falha reduz-se à probabilidade de nos enganarmos quanto ao número e tipo de falhas possíveis.

Best effort: para sistemas em que a segurança não seja crítica.

1.5.4 Resource adequate (or not)

O sistema que garante a resposta (granteed response) assenta no pressuposto de que existem recursos para responder às situações de picos de carga, isto é, situações anómalas (uma vez mais, uma questão de tolerância a falhas).

1.5.5 Event triggered vs time triggered

Em sistemas orientados no tempo os processos são desencadeados em instantes pré-determinados, enquanto que em sistemas orientados por eventos os processos são desencadeados quando existem mudanças nas variáveis de estado.

1.6 O mercado dos sistemas de tempo real

Uma das grandes aplicações destes sistemas são os chamados sistemas de tempo real embebidos. Estes sistemas fazem parte de um sistema mais abrangente denominado sistema inteligente. A crescente competitividade dos mercados exige produtos fiáveis a baixo custo. A utilização de sistemas de tempo real embebidos revela-se, cada vez mais, a opção que mais se adapta a esta exigência. Na mesma linha de raciocínio, sistemas de tempo real são cada vez mais utilizados na concepção de plantas industriais automatizadas, no intuito de reduzir os custos de produção.

Capítulo 2

Porquê uma solução distribuída ?

Uma arquitectura distribuída é composta por vários nós, e uma rede de comunicações que os interliga.
Para a implementação de sistemas de tempo real é preferível a utilização de uma arquitectura distribuída devido a alguns factores, que são apresentados nas secções seguintes.

2.1 Arquitectura do sistema

2.1.1 A forma segue a função

A função de um objecto deve determinar a sua forma física.
Num sistema distribuído, um nó é composto pela sua função lógica e pelo seu hardware. Visto num nível de abstracção maior, o nó pode ser analisado em termos das suas funções essenciais e propriedades temporais, escondendo-se assim, os detalhes irrelevantes da sua implementação física.
Em caso de falhas, a abstracção deve manter-se válida. Se se conseguir manter um mapeamento únivoco entre função e nó, facilmente se diagnostica uma falha, e podem-se simular que funções serão afectadas pelo mau funcionamento de um nó. Uma arquitectura distribuída habilita a aplicação do princípio *"A forma segue a função"*, para o projecto de sistemas computacionais.

2.1.2 Estrutura do *Hardware*

Uma aplicação pode ser dividida em *clusters*:

- Operacionais
- Computacionais
- Controlo de Objectos

Uma série de nós é interligada através de um interface de comunicação de tempo real. Um nó pode ser dividido em dois sistemas:

- Controlador de comunicações locais
- Computador

O interface entre estes dois sistemas é chamado de *communication-network interface* (*CNI*). O *CNI* está localizado na camada de transporte do modelo *OSI* e é considerado o mais importante interface numa arquitectura distribuída de tempo real.

2.1.3 O *Communication-Network Interface*

A função do sistema de comunicações de tempo real é transportar mensagens do *CNI* do nó emissor, para o *CNI* do nó receptor, num intervalo de tempo previsto, com baixa variação de latência e com alta reliabilidade. Deve ainda garantir a consistência das mensagens.

Semântica dos dados : No *CNI* do receptor existem dois tipos de mensagens:

Mensagens de Eventos - Têm de ser processadas uma só vez, por ordem temporal de acontecimento, e não por ordem de chegada (deve haver uma sincronização unívoca entre o receptor e o que envia).

Mensagens de Estado - Contendo informação sobre os vários sensores, p. ex., o receptor normalmente está interessado no último valor. A sua recepção não tem de ser processada numa estrutura do tipo *fila*. A semântica de mensagens de estado depende da natureza da aplicação de controlo.

Estratégia de Controlo : A decisão de quando uma mensagem deve ser enviada pode ser uma parte integrante do sistema de controlo *(controlo externo(*, ou do próprio sistema de comunicações *(controlo autónomo)*.

2.1.4 O Sistema de comunicações

Existem várias alternativas na implementação do sistema de comunicações.
A fiabilidade da comunicação pode ser aumentada quando se retransmite uma mensagem, ou através de replicação de mensagens.
Num sistema distribuído, o sistema de comunicações é um factor muito importante (deve ter uma reliabilidade várias ordens de grandeza superior do que os nós individuais), pelo que é analisado em maior detalhe no capítulo 7.

2.1.5 *Gateways*

O objectivo de uma *gateway* é interligar dois clusters. Tem a função de passar a informação relevante do *CNI* do emissor, para o *CNI* do receptor. Na maioria dos casos, apenas segmentos de informação de um cluster são relevantes para um outro cluster.

2.2 Composability

Sistemas grandes são, muitas vezes, construídos através da junção de vários subsistemas bem especificados e testados. É importante que as propriedades que foram estabelecidas nos subsistemas de mantenham constantes.

2.2.1 Definição

Uma arquitectura é dita *composable* a respeito de uma dada propriedade se a integração não invalida a propriedade definida no subsistema.

2.2.2 Sistemas de comunicação por eventos

Se um sistema de comunicação transporta mensagens de eventos, o controlo temporal é externo ao sistema de comunicação, sendo pertencente, ao sistema de controlo. Em sistemas de comunicação com o *meio partilhado*, tenta-se resolver conflitos de acesso, através de técnicas de acesso aleatório *(ethernet)*, ordens de acesso *(protocolos token)*, e prioridades nas mensagens *(CAN - Control Area Network)*.
Do ponto de vista temporal, sistemas de comunicação por eventos não são *composable*.

2.2.3 Sistemas de comunicação *Time-Triggered*

Neste tipo de comunicação, o controlo temporal reside no próprio sistema de comunicação. As propriedades temporais do *CNI* entre um nó e o sistema de comunicação ficam totalmente definidas na etapa de projecto. Dado que a integração não muda a especificações temporais do *CNI*, uma comunicação *time-triggered*, é *composable* no que diz respeito a propriedades temporais.

2.3 Scalability

Quase todos os sistemas envolvem ao longo do tempo. A evolução implica nos requerimentos ao sistema. Muitas funções tem de ser adicionadas e/ou modificadas ao longo do tempo
Um arquitectura escalável deve ser capaz de acompanhar estas alterações e não limitar de alguma forma a extensibilidade do sistema até um limite superior.

2.4 Extensibilidade

Uma arquitectura escalável não deve ter uma centralização limitada em capacidade de processamento ou capacidade de comunicação, sendo vantajoso a escolha de sistemas distribuídos.

2.5 Complexidade

Grandes sistemas podem ser construídos apenas se a sua complexidade for mantida sob controlo à medida que o sistema cresce.
O esforço necessário para perceber uma função particular deve manter-se constante e independente do tamanho do sistema.

2.6 Custo da silicone

a implementação de uma dada especificação com uma arquitectura distribuída requer inicialmente mais hardware do que uma arquitectura centralizada, devido ao sistema de comunicação, à função replicada de algumas funções, etc.
À medida que o tamanho de um sistema aumenta, existe um ponto em que um sistema centralizado se torna mais caro.
Um sistema centralizado pode trazer vantagens de custo a curto prazo, mas pode tornar-se desvantajoso a longo prazo.

2.7 Dependabilidade

Um sistema de tempo real requer a distribuição de funções de forma a se obter uma contenção de falhas, detecção de erros e tolerância a falhas, para que o serviço possa continuar independentemente da ocorrência de falhas. Se não pode ser demonstrado que as consequências de uma dada falha numa função não crítica não vão afectar as funções críticas, então a falha na função crítica deve também ser classificada de *catastrófica*.

2.7.1 Regiões de contenção a falhas

Um sistema tolerante a falhas deve estar estruturado em partições para que as consequências das falhas ocorridas numa partição possam ser detectadas e corrigidas sem que as suas consequências danifiquem ou influenciem o resto do sistema.
É difícil implementar-se este aspecto numa arquitectura centralizada, dado que muitas funções do sistemas são multiplexadas por vários serviços.

2.7.2 Replicação

Num sistema distribuído, um nó deve representar uma unidade de falha. Assim, as falhas podem ser menosprezadas através da inserção de nós replicadores.

2.7.3 Suporte à certificação

O projecto de sistemas de funções críticas, frequentemente tem de ser aprovado por uma agência de certificação independente. A certificação é dependente

da acumulação de provas analíticas e experimentais que garantam à entidade certificadora que o produto é indicado para a sua função.

2.8 Instalação física

É vantajosa a interacção física de um pequeno controlador num subsistema mecânico, de forma à construção de um componente com uma dada funcionalidade devido, nomeadamente, à diminuição dos erros devidos às cablagens e conexões, à autonomia, e à facilidade de instalação.

Capítulo 3

Tempo Global

Toda a discussão é feita com base no estabelecimento de regras que permitam a ordenação de eventos no tempo.

3.1 Tempo e ordem

Tipos de ordenação

Ordem temporal: etiquetação de um dado evento em termos do instante de ocorrência.

Ordem causal: O desvio de uma variável implica normalmente o desvio de outras variáveis com ela relacionadas. O estabelecimento de uma ordem causal serve para que seja possível depreender relações de causa/efeito.

3.1.1 Relógios

Relógio físico: dispositivo para medir o tempo. Consiste num contador a que se associa um mecanismo de oscilação para gerar periodicamente um evento que incremente o contador. Esse evento periódico chama-se microtick.

Granularidade: Intervalo de tempo (duração) entre dois microticks consecutivos de um dado relógio.

Relógio de referência: relógio cujo contador é sempre igual ao ditado pelo *International Time Standard*. Este relógio tem frequência f_z e granularidade $1/f_z =$ período de oscilação (logo, duração entre dois microticks).

Granularidade de um relógio k: número de microticks do relógio de referência entre dois microticks do relógio k.

Offset: dados dois relógios, é a diferença entre dois microticks idênticos desses relógios.

$$Offset = |z(microtick_i^j) - z(microtick_i^k)|$$

3.1.2 Precisão e exactidão

Precisão: máximo offset entre um conjunto de relógios no conjunto de todos os offsets desse conjunto.

Exactidão: precisão do relógio k. No fundo, e a precisão do relógio k em relação ao relógio de referência.

Sincronização interna: sincronização mútua que é feita tomando como base o valor de todos os relógios do conjunto.

Sincronização externa: sincronização que é feita recorrendo ao valor fornecido por um relógio de referência.

3.1.3 Referências padrão

Temps atomique internationale (IAT): definição laboratorial do segundo como a duração de 9192631770 períodos da radiação de transição do átomo cesium 133. A intenção foi obter um valor mesurável (em laboratório) e bem definido, e que concordasse o mais possível com a definição *UTC*.

Universal Time Coordinated: esta definição é baseada no período de rotação da terra em torno do sol. Como não existe igualdade entre as duas referências, a base de tempo *UTC* tem de ser periodicamente sincronizada com a *IAT*.

3.2 Medir o tempo

Problema: dado um sistema distribuído, com cada nó a possuir um relógio autónomo, como verificar se dois eventos podem ser cronologicamente ordenados quando são verificados por nós diferentes? A solução é sincronização de relógios.

Tempo global: noção que pressupõe o estabelecimento de um método que consiste em dispoletar periodicamente a sincronização dos relógios do sistema distribuído.

Os fundamentos para a implementação estão no livro. Seria ambicioso estar aqui a traduzir. No entanto, a grande conclusão a tirar dessa apresentação é que dois eventos só podem ser cronologicamente ordenados em termos da base de tempo global se estiverem separados por mais de dois granulos (2g). Este é o limite fundamental para a ordenação de eventos.
Este fenómeno não deixa de ser quase intuitivo, no entanto. Existe um erro associado à digitalização do instante que é medido (o tempo físico é uma entidade, por essência, contínua). Esse erro nunca será menor que a granularidade do relógio de referência. A esse erro há que somar a precisão do conjunto de todos os relógios do sistema que será também, no máximo, igual à granularidade do relógio de referência.

3.3 Sincronização interna

Uma dada função de convergência ”acerta” periodicamente os relógios de modo a permitir que estes se mantenham sempre dentro de uma precisão especificada. Apresentam-se a seguir dois métodos de implementação interna.

Central Master Syncronization: um mestre envia aos seus servos o tempo de referência. Tem o problema de sofrer com a falha do mestre, caso ela ocorra. Pode ser resolvido por replicação.

Distributed Syncronization: a função de convergência toma referências de todos os relógios do sistema distribuido e estima o valor actual. Como corre localmente e "vê"o mesmo que os outros vêem, o valor retornado pela função é igual para todos.

3.4 Sincronização externa

Um pouco à semelhança do método Central Master Syncronization, um servidor de carácter "universal"envia periodicamente a referência "universal". Entre os muitos protocolos que implementam este tipo de sincronização encontra-se o Network Time Protocol, que é o protocolo utilizado pela Internet.
O formato do tempo proposto por este protocolo consiste em oito bytes divididos em dois campos: os primeiros quatro bytes representam a parte inteira do número de segundos contados desde as 0h00 do dia 1 de Janeiro de 1900, segundo o padrão UTC. Os segundos 4 bytes reprentam a parte decimal com uma resolução de 232 picosegundos, aproximadamente.

Capítulo 4

Modelização de sistemas de tempo real

Neste capítulo, é desenvolvido um modelo conceptual de um sistema de tempo real distribuído.
O foco é feito na estrutura do sistema e os aspectos temporais.

4.1 Abstracções Correctas

4.1.1 O objectivo do modelo

Um modelo que introduz um conjunto de conceitos bem definidos e as suas inter relações é chamado de *modelo conceptual*.
Nesta secção desenvolve-se um modelo conceptual para perceber o comportamento temporal de um sistema computacional de tempo real.
A probabilidade que as hipóteses feitas no processo de construção do modelo se mantenham válidas na realidade é chamada de *assumption coverage* (limita a probabilidade de conclusões derivadas do modelo sejam válidas no mundo real).
Duas suposições existem quando se projecta um modelo de um sistema de projecta um modelo de um sistema de tempo real tolerante a falhas:

Suposição de Carga - Existe uma capacidade de processamento limitada. A resposta temporal de um sistema só pode ser analisada quando o sistema está abaixo da sua carga máxima *(peak load)*.

Suposição de Falha - O sistema de tempo real tolerante a falhas , é projectado para tolerar todas as falhas previstas, i.e., se a falta que ocorre no mundo real não for prevista pela Suposição de Falha, o sistema de tempo real tolerante a falhas pode não ser válido.

4.1.2 O que é relevante ?

Neta secção discutem-se quais as propriedades temporais no mundo real que devem fazer parte do modelo de um sistema distribuído de tempo real.

Tempo Físico. A progressão do tempo físico é bastante importante num sistema distribuído de tempo real. Assume-se que o observador externo, com a referência de relógio "z"está presente e todos os relógios de todos os nós estão sincronizados com uma precisão "π", que é suficiente para a aplicação.

Duração das acções. A execução de uma ordem é uma acção. Dada uma acção "a", distinguem-se as seguintes quantidades,

Duração actual dada uma entrada concreta "x", denota-se por $d_{act}(a, x)$, o número de unidades de tempo do relógio de referência "z"que ocorrem entre o início e o fim da acção.

Duração mínima $d_{min}(a)$ é o mais pequeno intervalo de tempo que leva a completar a acção "a", quantificada por todos os dados de entrada disponíveis.

Tempo de execução no pior caso (*WCET - Worst Case Execution Time*) $d_{WCET}(a)$ é o maior intervalo de tempo que leva a completar a acção "a", quantificada por todos os dados de entrada disponíveis.

jitter É a diferença entre o $d_{WCET}(a)$ e $d_{min}(a)$.

Frequência de activações. O máximo número de activações de uma acção por unidade de tempo é chamado de *frequência de activação*.
Um recurso só pode cumprir a sua especificação de tempo real se a frequência e a distribuição temporal das activações forem estritamente controladas.

4.1.3 O que é irrelevante ?

Dado que o modelo é uma interpretação reduzida da realidade, é importante definir os parâmetros irrelevantes dado que a sua introdução no modelo complica a representação e a análise do problema desnecessariamente.

Pormenores de Representação. O objectivo de um modelo conceptual de um sistema de tempo real é nas propriedades temporais e no significado das variáveis no tempo real e não na sua aparência sintáctica.

Detalhes da transformação dos dados. Num sistema de tempo real existem programas que calculam um resultado mediante dados de entrada.
Parâmetros como a lógica interna do programa e os resultados intermédios são irrelevantes no domínio do modelo conceptual.

4.2 Elementos Estruturais

Uma aplicação distribuída de tempo real tolerante a falhas pode ser decomposta num conjunto de clusters que comunicam entre si, que por sua vez podem ser decompostos num conjunto de unidades tolerantes a falhas *(FTU)*, ligadas por uma rede de tempo real.
Cada *FTU* consiste num ou mais nós. Dentro de cada nó estão a ser executadas concorrentemente tarefas que satisfazem uma dada função.
Explicam-se em seguida estes blocos.

4.2.1 Tarefa

Uma tarefa é a execução sequencial de um dado programa. A ordem de início desta tarefa é fornecida pelo sistema operativo.

Tarefa simples - Uma rotina sem pontos de sincronização, i.e., desde que começa pode ir até ao fim. O tempo de execução deste tipo de tarefas não depende directamente do progresso das outras tarefas. Depende directamente no caso de preempção de tarefas quando é interrompida por uma outra tarefa de prioridade superior.

Tarefa complexa - Quando contém instruções de sincronização com outras tarefas. Assim, o $WCET$ de uma tarefa complexa é um problema global, dada a dependência directa do progresso das outras tarefas.

4.2.2 Nó

É um bloco computacional com o seu próprio *hardware* e *software* que executa um dado conjunto de funções no seio de um sistema computacional distribuído.
Do ponto de vista da rede, a função e temporização de um nó é caracterizado pelas mensagens que envia e recebe do meio de comunicação.

Estrutura do nó. O hardware de um nó consiste num computador, *CNI* e controlador de comunicações. Por sua vez, o computador é composto por um *CPU*, memória, e um relógio de tempo real sincronizado.
O *CNI* é partilhado pelo computador e pelo controlador de comunicações.
Em muitas aplicações o nó é a mais pequena unidade que pode ser substituída no caso de uma falha.

4.2.3 Unidade Tolerante a Falhas

A unidade é uma abstracção que é introduzida para implementar a tolerância a falhas através da replicação activa. Consiste numa série de nós replicados que produzem uma réplica determinada de mensagens de resultados.

4.2.4 Cluster computacional

Um cluster computacional é uma série de unidades tolerantes a falhas que cooperam de forma a resolver o serviço tolerante a falhas para os serviços do cluster.

4.3 Interfaces

Uma actividade importante no projecto de arquitecturas de sistemas de tempo real é o projecto de *interfaces*. Um *interface* é a ligação comum entre dois subsistemas.
Um *interface* entre dois subsistemas de um sistema distribuído de tempo real pode ser caracterizado por :

- As propriedades de controlo, i.e., as propriedades dos sinais de controlo que passam pelo interface.
- As propriedades temporais, i.e., as restrições temporais que têm de ser satisfeitas pelos sinais de controlo e pelos dados que passam no *interface*.
- A função pretendida, i.e., a especificação da função pretendida no interface do receptor/emissor.
- As propriedades dos dados, i.e., a estrutura e semântica dos elementos de dados que passam no interface.

4.3.1 Obrigações temporais de clientes e servidores

Num modelo cliente-servidor, um pedido (mensagem) de um cliente para o servidor origina uma resposta do servidor instantes depois. Essa resposta pode ser uma mudança de estado do servidor e/ou a transmissão de uma resposta (mensagem) ao cliente.
Existem três parâmetros que caracterizam essa interacção:

- O tempo de resposta máximo, $RESP$, que é esperado pelo cliente; encontra-se especificado.
- O $WCET$ do servidor que é determinado pela implementação do servidor.
- O tempo mínimo $MINT$ entre dois pedidos consecutivos do cliente.

Num sistema de tempo real *duro*, a implementação deve garantir que a condição:

$$WCET < RESP$$

é válida desde que o cliente respeite a obrigação de esperar $MINT$ entre pedidos.

4.4 Controlo temporal vs. Controlo lógico

O controlo lógico apenas se encontra interessado no controlo lógico de uma tarefa, que é determinada pela estrutura do programa e pelos dados de entrada, de forma a obter a transformação de dados desejada.

O controlo temporal implementa a determinação dos instantes de tempo quando uma tarefa deve ser activada, ou bloqueada, dado que algumas condições externas à tarefa não são satisfeitas num dado instante.

Num sistema activado pelo tempo, o controlo mantém-se sempre interno ao sistema computacional. De forma a reconhecer as alterações fora do sistema computacional, um sistema activado

4.4.1 *Event Triggered* vs. *Time-Triggered*

Um sinal de controlo temporal para a activação de uma tarefa num nó pode activar-se de uma das duas fontes seguintes:

- Um sinal de controlo é derivado de uma mudança de estado, um evento, no ambiente ou no sistema de computação.
- Um sinal de controlo é derivado da progressão do tempo real. Sempre que o relógio de tempo real num nó chegue a um valor especificado numa tabela de escalonamento, um sinal de controlo temporal é gerado.

4.4.2 Interrupções

Num sistema accionado por eventos, os accionamentos externos são comunicados ao sistema computacional através de um mecanismo de interrupções.
Uma interrupção é um pedido assíncrono (suportado por hardware) para a activação de uma tarefa específica causada por um evento externo para a computação actual.

***Custo* de uma interrupção** Quando um mecanismo de uma interrupção é activado e uma interrupção ocorre, a execução da tarefa actual é parada e são guardados os valores computacionais da tarefa actual.
Estas comutações introduzem um tempo acrescido na execução das tarefas ($WCAO$ - *Worst-Case administrative Overhead*) que corresponde a um menor tempo útil do processador para a execução de tarefas mesmo que as interrupções tenham sido geradas erradamente.

4.4.3 Tarefa por temporização

Num sistema activado pelo tempo, o controlo mantém-se sempre interno ao sistema computacional. De forma a reconhecer as alterações fora do sistema computacional, um sistema activado pelo tempo deve periodicamente verificar o estado do ambiente através de uma tarefa de amostragem. Dado que os estados do sistema, quer internos, quer externos, são adquiridos à frequência da tarefa de amostragem, apenas os sinais com uma duração maior que o período da tarefa de amostragem podem ser garantidos de serem amostrado.

Custo de uma tarefa temporizada A tarefa de amostragem periódica gera um custo num sistema controlado por tempo. O período da tarefa de amostragem deve ser menor que a diferença entre o fim do prazo de execução *(deadline)* e o tempo de execução.

4.5 Tempo de execução no pior caso *(WCET)*

O tempo limite para completar uma transacção de tempo real apenas pode ser garantido se o *WCE* de todas as tarefas da aplicação que são partes integrantes da transacção forem conhecidos "à-priori".
O *WCET* duma tarefa é o limite superior de tempo entre o início e o fim de uma tarefa. Deve ser válido para todas as combinações de entrada e cenários de execução da tarefa. Além do *WCET* das tarefas da aplicação é importante saber os tempos de atraso causados por tarefas administrativas *(WCAO - Worst-Case Administrative Overhead)*

4.5.1 *WCET* de tarefas simples

O *WCET* de tarefas simples depende de:

O código fonte da tarefa. Para a análise do *WCET* dum programa escrito numa linguagem de alto nível, deve ser analisada qual a sequência de instruções a ser executadas no pior cenário. O caminho maior (mais demorado) num programa é chamado de *"caminho crítico"*.

As propriedades do código gerado pelo compilador. Partindo do princípio que são conhecidos os tempos de execução máximos dos comandos em linguagem máquina, pode-se analisar a estratégia de geração de código do compilador. Pode ser gerada uma *"árvore de temporizações"* durante a compilação. Os efeitos de alocação de registos, optimização de código, e outras opções tomadas no processo de compilação devem ser consideradas na análise do *WCET*.

As características do hardware onde a tarefa vai ser executada. Caso o hardware utilizado tenha tempos fixos de execução de instruções pode ser consultadas as especificações do fabricante. No entanto, em arquitecturas como a *RISC*, tem de ser ter em conta o *pipeline* e a cache de instruções e de dados, o que não se torna uma tarefa fácil. É, no entanto, possível analisar estatisticamente o *WCET* duma dada instrução num dado programa duma dada arquitectura, recorrendo à análise dos segmentos de código e tendo em conta o pior cenário.

4.5.2 Tarefas simples preemptíveis

Quando uma tarefa simples é interrompida por outra tarefa coma prioridade superior, a tarefa inicial é estendida pelos seguintes factores :

- O *WCET* da tarefa que a interrompeu
- O *WCET* do sistema operativo

- O tempo necessário para repor as caches de instruções de dados sempre que o contexto do processador é comutado.

À soma dos tempos causados pelo sistema operativo e o refrescamento das caches é normalmente chamado de *WCAO* (*Worst-Case Administrative Overhead*).
O *WCAO* é um peso não producente que seria evitado se a preempção de tarefas fosse proibida.

4.5.3 *WCET* de tarefas complexas

O *WCET* de tarefas complexas depende não só da performance da tarefa em si, mas também do comportamento das outras tarefas e do sistema operativo, dentro de um nó.
Além da preempção de tarefas analisadas na secção anterior, é preciso ter em conta as dependências das tarefas (exclusão mútua, precedência, etc.).

4.5.4 Na prática

Actualmente, a análise sistemática de todos os efeitos de forma a determinar o *WCET* duma tarefa complexa ainda se encontra numa fase inicial.
No entanto, o *WCET* pode ser estimado na prática combinando diversas técnicas:

- A medida dos tempos numa implementação de forma a ganhar dados para um *WCET* experimental.
- A utilização de uma arquitectura restrita que reduza as interacções entre as tarefas e facilite uma análise *à-priori* da estrutura de controlo.
- A análise de sub problemas (análise do código fonte) de forma a gerar mecanicamente testes passíveis de se obter uma aproximação do *WCET*.
- Testes exaustivos da implementação completa para validar as hipóteses de forma a medir as margens do *WCET* assumido com as medidas actuais.

4.6 O historial de estados

O historial de estados num qualquer ponto de interrupção pode ser definido como os conteúdos do *program counter* e todas as estruturas de dados que teriam de ser carregadas num dispositivo "virgem" para que fosse possível resumir a operação no ponto da interrupção.
O tamanho do historial de estados depende do nível de abstracção e do instante temporal de observação do sistema. Se a granularidade das observações é aumentada e se o instante de observação é escolhido imediatamente antes ou a seguir a uma operação atómica num dado nível de abstracção, o tamanho do historial de estado pode ser reduzido. Um historial de estado pequeno simplifica a reintegração de um componente com falhas no processo de reintegração.

4.6.1 *Ground State*

O conceito de *ground state* dum nó num sistema num dado nível de abstracção pode ser definido como um estado em que nenhuma tarefa se encontra activa e onde todas as filas de espera na comunicação se encontram vazias. Se um nó se encontra no *ground state*, então o historial de estados do nó é limitado às estruturas de dados visíveis e ao *program counter*. A reintegração de um nó após uma falha é simplificada se um nó periodicamente se encontra num *ground state* que pode ser utilizado com um ponto de reintegração.

Capítulo 5

Entidades e Imagens de Tempo Real

Neste capítulo abordam-se os conceitos de entidades, objectos e imagens de Tempo Real (TR), estimação de estado (prediction) e procedimentos para a sua obtenção, a dependência entre objectos de TR e relógios de TR. Action Delay, observações paramétricas e phase sensitive de entidades TR. Imagens TR e exactidão temporal. Finalmente, a exposição do conceito de replicação determinística para tolerância a falhas.

5.1 Entidades TR

São variáveis com atributos imutáveis: nome, tipo, domínio de valor e valor máximo da taxa de variação. Têm também atributos dinâmicos: valor instantâneo e taxa de variação instantânea (derivada). São as variáveis que interessa monitorizar ou controlar num dado sistema.

5.1.1 Esfera (ou Âmbito) de Controlo

Uma dada entidade (variável) classifica-se como observável ou controlável conforme o sistema onde essa questão se coloca. Por exemplo, um condutor não controla directamente a velocidade mas pode observá-la.

5.1.2 Variáveis TR discretas e contínuas

Distinguem-se pela gama de valores que podem assumir, conforme esta seja discreta ou contínua.

5.2 Observação

É uma etiqueta formada pelos 3 atributos seguintes: nome, instante de observação e valor.
Importante: em variáveis discretas, o valor da variável não muda instantaneamente; é preciso esperar que o valor estabilize até que a observação possa ser efectuada.

5.2.1 Observação Intemporal

Em sistemas sem base de tempo global, uma mensagem que é enviada sobre o instante de uma observação de uma variável TR não tem significado para o receptor pois não existe maneira de comparar instantes de tempo.

5.2.2 Observação Indirecta

Quando uma variável não pode ser medida directamente torna-se necessário estimar um valor. Esta estimação é feita recorrendo a um modelo do sistema.

5.2.3 Observação de Estado

Uma observação chama-se observação de estado se o atributo (ou campo) "valor"representar o estado da variável RT.

5.2.4 Observação de evento

Quando estamos em presença de uma observação de evento, o campo "valor"é a diferença entre o novo valor e o valor antigo.
Problemas associados:

- Este tipo de observação é feita por eventos periódicos que a despoletam. Assim, o instante de observação não é conhecido com precisão a não ser a precisão dada pelo período de observação.

- O extravio ou perda de uma mensagem, pode ser interpretado como uma não variável. Por este motivo, observações de estado são mais frágeis na tolerância a falhas.

5.3 Imagens e Objectos de Tempo Real

5.3.1 Imagens TR

É como uma observação de uma variável mas, ao contrário desta, não tem associado o instante de captura da variável. Tem um "prazo de validade".

5.3.2 Objectos TR

É como um contentor que tem guardada a imagem, ou mesmo a pória observação da entidade de Tempo Real.

Objectos TR distribuidos : réplicas do objecto são distribuídas pelo sistema para implementar tolerância a falhas.

5.3.3 Exemplos

Tempo global - Ver Capítulo 3.

Membership Service - Num dado instante é gerada uma mensagem sobre o estado de todos os nós do sistema e é enviada a todos os nós.

5.4 Exactidão Temporal

5.4.1 Definição

É uma relação temporal entre a variável RT e a imagem (ou objecto) dessa variável. É o valor da imagem que coincide com uma das últimas observações.

Intervalo da Exactidão. É a duração entre o instante de referência à imagem e o instante em que a variável foi observada.

Para que serve? Existe um erro associado à utilização da imagem como "cópia" da variável. Esse erro é igual à taxa de variação da variável multiplicado pelo intervalo de exactidão (dacc). Casos em que a variável mude rapidamente requerem um dacc muito pequeno. Isto significa que o tempo entre a observação da variável e o uso da imagem tem que ser menor que dacc.

Transacção de "fase sincronizada". É uma transacção composta pelas seguintes três etapas: envio, comunicação e recepção. Cada uma destas etapas começa imediatamente após acabar a anterior. O WCET desta transacção é a soma dos 3 individuais. Esta soma tem que ser menor que dacc, caso contrário torna-se necessário fazer uma *estimação de estado*.

5.4.2 Classificação de Imagens de Tempo Real

Phase Insensitive Dado um determinado período de observação, se imagem é "phase insensitive" se:

$$d_{acc} > (d_{update} + WCET_{send} + WCCOM + WCET_{rec})$$

Phase sensitive No caso em que

$$d_{acc} < d_{update} + WCET_{send} + WCCOM + WCET_{rec}$$

e

$$d_{acc} > WCET_{send} + WCCOM + WCET_{rec}$$

Uma imagem que é phase sensitive torna o escalonamento da tarefa que a usa mais complicado. Este problema pode ser minorado quando se pode diminuir dupdate ou, caso não seja possível, quando se poder fazer uma estimação de estado. Mas atenção:

- se d_{update} diminui - aumenta volume de comunicações
- se estimação de estado - aumento da carga no processador

É preciso decidir!

5.4.3 Estimação de Estado

Para se estimar um estado duma entidade TR num ponto futuro é preciso criar um modelo da entidade, de modo a fazer a estimação periodicamente dentro do objecto que detém essa entidade. O instante para o qual a imagem deve ser estimada é aquele em que se tornar necessário usar a imagem. A estimação de estado prolonga o intervalo de exactidão, mas apenas se pode aplicar a processos determinísticos, onde não exista alguma espécie de desempenho aleatório.

Input do modelo de estimação de estado. O mais importante é ter uma base de tempo com uma boa precisão dado que, num sistema distribuído, a observação e o uso da imagem são feitos em pontos distintos do sistema. A prática mais comum é, sabendo a primeira derivada (admitindo que se construiu um modelo contínuo e diferenciável), calcular a estimativa com base numa expansão em série, tipo série de Taylor; o uso de modelos matemáticos mais detalhados, quando necessário, conduz a um maior consumo de recursos!

5.4.4 Uma vez mais "Composability"

Uma dada variável é lida e uma mensagem referente é enviada. Tempos diferentes na propagação da mensagem (seja lá de onde for que ocorreu a diferença de tempo de execução) conduzem a intervalos de estimação de estados diferentes. Então, o que se costuma fazer são duas estimativas separadas.

Figura 5.1: Latência no emissor e no receptor

5.5 Permanência e Idempotência

5.5.1 Permanência

Uma mensagem que chega a um nó torna-se permanente quando o nó souber do paradeiro de todas as mensagens anteriores a ela. Por exemplo, na monotorização de alarmes, um alarme só pode ser despoletado depois de o "monitor"ter recebido todo um conjunto de informações que tem implicação no alarme. Por outras palavras, só quando a mensagem de alarme se tornar permanente, pois pode ter existido alguma mensagem prévia que anule o efeito da activação do alarme.

Delay de actuação: intervalo entre o começo do envio de uma mensagem e o instante em que ele se torna permanente no receptor. Nenhuma acção deve ser inicializada até passar o intervalo para evitar um comportamento errado.

Acção irrevogável: acção que uma vez despoletada se processa até ao fim. É necessário ter especial atenção nestas acções quanto ao Delay de actuação.

Duração do Delay de uma acção: Depende do *jitter* de comunicação $(d_{max} - d_{min})$ e da rapidez com que o sistema se "apercebe" dos eventos.

Sistemas com tempo global: o instante em que a mensagem é enviada pode fazer parte da mensagem o instante em que a mensagem se vai tornar permanente é

$$t_{perm} = t_{send} + d_{max} \cdot (\text{delay de comunicação}) + 2g$$

com g sendo a granularidade da base de tempo (o Capítulo 3 mostra o porquê de $2g$).

Sistemas sem tempo global. O instante t_send não é conhecido, logo o receptor tem que esperar que o *jitter* associado ao desvio da a mensagem se esgote. Na realidade, a mensagem pode já ter estado d_max no processo de transmissão. Então, o tempo que a mensagem demorará a ficar permanente vai ser

$$T_{permanent} = t_{send} + d_{max} - d_{min} + gl$$

Com gl sendo a granularidade da base de tempo local. Sistemas sem base de tempo global são mais lentos.

Intervalo de exactidão vs delay de uma acção. Uma imagem só pode ser utilizada se a mensagem for permanente e se a mensagem estiver temporalmente exacta. Sem estimação de estado, esta condição só se verifica no intervalo [t_{perm}, t_{obs}, t_{dacc}]. Enquanto que d_{acc} depende da dinâmica do sistema de controlo. (t_{perm} - t_{obs}) depende da implementação. Se a implementação não satisfaz os requisitos temporais, então a única alternativa é estimação de estado.

5.5.2 Idempotência

Se o resultado de um nó receber mais do que uma réplica de mensagens é o mesmo que se recebesse apenas uma, então o conjunto das réplicas é idempotente e a implementação de tolerância a falhas por via de mensagens replicadas é facilitado.

5.6 Replicação determinística

É uma relação que se estabelece entre um conjunto de objectos ou um conjunto de nós.

Entre objectos: todos os objectos do conjunto tem o mesmo *h-state* no ponto de ground-state, e vê que produzem as mesmas mensagens em instantes que não diferem mais do que x unidades de tempo, para quem os observa.

Entre nós: quem os observa vê o mesmo *h-state* no ponto de *ground-state*, e vê que produzem as mesmas mensagens em instantes que não diferem mais do que x unidades de tempo.

x: é o tempo necessário para corrigir uma mensagem errada (ou falta de mensagem) num conjunto de réplicas redundantes.

Para que serve replicação determinística?

- Implementar tolerância a falhas por redundância num conjunto de nós ou objectos replicados. Esta implementação assenta na ideia de ”maioria de voto”.
- Facilita o teste ao sistema.

5.6.1 Ponto Fulcral de Decisão

É um ponto de decisão num determinado algorítmo que apresenta várias alternativas possíveis. É importante porque nós replicados devem seguir as mesmas ”trajectórias”.

5.6.2 Causas principais de replicação não-determinística

Inputs Divergentes. São as que resultam da digitalização (e discretização) de sinais contínuos. O tempo, visto como conjunto contínuo (denso), também é afectado por este fenómeno. É inevitável!

5.6.3 Implementar um sistema com replicação determinística

De seguida apresentam-se os fundamentos para construir um sistema com replicação determinística de modo a evitar (ou prevenir) as situações descritas anteriormente.

Bases de tempo discretas: isto implica que não se irá impor time-out’s para nenhum processo relevante, pois a noção de intervalo de tempo é contínua.

Consistência de dados (agreement ou input): estabelecer um protocolo que define um instante de ocorrências de eventos que seja igual para todas as réplicas.

Estruturas de controlo estatísticas: implementar estruturas de controlo cuja validação não dependa dos dados.

Algorítmos determinísticos: no fundo assenta na ideia de evitar construções cujo o resultado da operação não estrja bem definido como no caso da instrução wait (race conditions).

5.6.4 Protocolo Leader-Follower

Assenta no conceito de uma seita: um líder com o seus seguidores. Requer muita coordenação entre as réplicas.

- Pontos a favor: reduz o espaço restrito apresentado anteriormente para o estabelecimento de replicação determinística.

- Pontos contra: reduz a independência entre as réplicas e, logo, o próprio conceito de replicação aplicada à contenção de erros. Por outro lado, a coordenação exige grande fluxo de comunicações, pelo que é necessário uma maior largura de banda. A resposta ao aumento do fluxo nos canais de comunicação é um aumento dos tempos de processamento.

Desvios entre o progresso do processamento e o tempo "físico": muitos CPU'S executam um algorítmo de correcção quando há ocorrência de erros inesperados no processamento de instruções. Como a ocorrência destes erros é não-determinística, diferenças entre o progresso do tempo e o processamento de dados são enevitáveis entre nós replicados. No limite, conseguir-se-á estabelecer uma probabilidade de ocorrência desse tipo de erro mas nunca se elimina o factor aleatório.

Deriva entre osciladores: dois osciladores *nunca* poderão ter a mesma frequência de oscilação. É um fenómeno que resulta da sua concepção física.

Escalonamento preemptivo: provoca o reconhecimento de eventos em instantes distintos.

Características não determinísticas da linguagem: em linguagens onde a decisão, num dado ponto de decisão, não é conhecida à partida.

Race conditions: em sistemas que competem por recursos e cujo o algorítmo de atribuição é não determinístico.

Capítulo 6

Tolerância a Falhas

A tolerância a falhas é importante em sistemas criticos de tempo real dado que uma simples falha num componente poderia levar a uma falha catastrófica no sistema. A detecção de erros requer o conhecimento acerca do correcto funcionamento do sistema

6.1 Falhas, erros e faltas

Sempre que um serviço de um sistema, visto pelo utilizador do sistema, desvia-se da especificação do sistema, diz-se que o sistema *falhou*

6.1.1 Falhas

Um a falha é um elemento que denota um desvio entre o serviço actual e o serviço especificado ou pretendido, ocorrendo num ponto particular em tempo real. As seguintes classificações podem ser efectuadas:

Natureza da falha - *Em Valor* - Uma falha em valor significa que um valor incorrecto é apresentado ao utilizador do sistema.
No Tempo - Uma falha no tempo significa que um valor é apresentado ao utilizador do sistema num instante fora do especificado.

Percepção da falha - *Consistente* - Num sistema com vários utilizadores todos vêm o mesmo valor (possivelmente errado). Se um sistema produz resultados correctos ou nenhum resultado, é chamada de falha *fail-silent*. Se o sistema deixa de operar depois da primeira falha *fail-silent*, a falha é chamada de *crash-failure*. Uma *crash-failure* que é conhecida em todo o sistema é chamada de falha *fail-stop*.
Inconsistente - Numa situação de falha inconsistente, diferentes utilizadores podem receber diferentes resultados. Um subsistema malicioso pode perturbar o sistema de operações correctos mostrando-lhe diferentes e contraditórias faces de uma falha. Falhas inconsistentes são também chamadas *two-faced*, *malicious* ou *byzantine*. Para tolerar uma determinada falha k são necessários:

- $k + 1$ componentes se as falhas forem *fail-safe*
- $2k + 1$ componentes se as falhas forem *fail-consistent*
- $3k + 1$ componentes se as falhas forem *malicious*.

Efeito da falha - *Benigno* - Apenas pode causar custos de falhas na mesma ordem de amplitude como a perda do sistema.
- *Maligno* - Pode causar custos de falhas várias vezes a ordem de amplitude como a perda do sistema. A aplicações onde falhas malignas podem ocorrer denominam-se aplicações *safe-critical*.

Periodicidade Num dado intervalo de tempo, uma falha pode ocorrer uma ou várias vezes. Se ocorrer uma única vez é chamada de falha única. Um caso especial é a falha permanente em que o sistema deixa de funcionar até ser retirada a causa da falha. Se o sistema continua a operar depois da falha, chama-se à falha *transiente*. Uma falha que ocorra várias vezes é chamada de *intermitente*. As razões mais usuais para uma falha permanente num dado sistema, são o número de pinos e o empacotamento de um dado circuito integrado assim como as cablagens. Como causas para uma falha transiente temos a interferência electromagnética, perturbações nas fontes de alimentação e particular de alta energia.

6.1.2 Erros

A maioria das falhas pode ser rastreada para um estado interno incorrecto no computador. Tal estado interno incorrecto é chamado de *erro*.
Um erro *transiente* é um erro que existe apenas durante um curto espaço de tempo e depois desaparece sem uma acção explícita de reparação. Se o erro persiste até uma acção de reparação o erro é chamado de erro *permanente*.
Numa arquitectura tolerante a falhas, cada erro deve ser confinado a uma só região para evitar a propagação do erro através do sistema.

6.1.3 Faltas

A causa do erro, logo a causa indirecta de uma falha é chamada de *falta*. as faltas podem ser caracterizadas da seguinte maneira. ...

6.1.4 Tolerância a falhas sistemáticas vs. especificas da aplicação

Nenhum sistema complexo sobreviverá por um período estendido de tempo sem tolerância a falhas. Existem duas hipóteses de implementação :

Sistemática - No nível arquitectural, transparente ao código da aplicação. A arquitectura deve fornecer uma réplica determinista para que a tolerância a falhas possa ser atingida através da replicação temporal ou espacial das computações, de forma a detectar e eliminar as faltas especificadas na hipótese de falhas.

No nível da Aplicação, dentro do código da aplicação.

6.2 Detecção de erros

Um erro é a discrepância entre o estado desejado e o correcto de um dado sistema.
O objectivo da computação tolerante a falhas é detectar e ultrapassar ou corrigir os erros antes que eles se manifestem como falhas na interface do utilizador.
A detecção de erros necessita, que, além da informação acerca do estado actual do sistema, seja dada informação acerca do estado correcto do sistema no instante seguinte, quer através de um conhecimento *à-priori*, ou através da comparação dos resultados de dois ou mais canais redundantes de computação.

6.2.1 Baseada num conhecimento *à-priori*

Quanto mais informações estão disponíveis acerca das propriedades dos estados correctos e dos padrões temporais de uma computação correcta, mais efectivas se tornam as técnicas de detecção de erros.
Algumas técnicas de detecção de erros:

- Conhecimento da sintaxe no espaço de código
- Testes de plausibilidade
- Padrões de activação de computação
- Utilização do *WCET* duma tarefa

6.2.2 Baseada em computações redundantes

Existem várias combinações de hardware, software e redundância de tempo que podem ser utilizadas para detectar certos tipos de erros através de uma computação duplicada.
Esta técnica pode ser aplicada para aumentar a cobertura de detecção de erros, mesmo que não possa ser garantido que todas as tarefas iniciadas consigam ser completadas duas vezes no intervalo de tempo disponível.

6.3 O nó como unidade de falha

O nó é uma unidade contida que fornece uma função através de uma interface definida.
No nível arquitectural, é considerado o comportamento de um nó. Neste nível o nó deve exibir apenas falhas *fail-silent*. Neste caso os mecanismos de tolerância a falhas devem executar duas tarefas:

- detectar a falha do nó e reportar esse estado a todos os nós numa baixa latência.
- Mascarar a falha do nó através da redundância activa e reintegrar os nós reparados assim que estes estão disponíveis.

6.4 Nível mínimo de um nó

Grandes sistemas têm vários estados entre completamente operacional e não operacional. As especificações do sistema devem conter directivas acerca do nível de serviço mínimo de cada uma das funções do sistema.

6.4.1 Detecção de erros num nó

Um nó deve detectar todas as falhas internas com uma baixa latência, e deve reportar essas falhas através de uma única falha externa *fail-silent*. A detecção de erros no nó deve ser feita no nível de falhas de valores e especificações temporais.

6.4.2 Suporte a excepções

As excepções são uma técnica bem conhecida para o manuseamento de erros que são detectados dentro de uma tarefa. Depois do levantamento de uma excepção, quer por software ou por hardware, o controlo é transferido para um sistema de manuseamento de interrupções. Depois deste sistema ter terminado, o controlo é resumido no ponto onde a excepção ocorreu ou a tarefa é terminada.
Em sistemas de tempo real as excepções devem ser tidas em conta dado que o $WCET$ de uma tarefa é aumentado pelo $WCET$ de cada excepção que possa ocorrer.

6.5 Unidades de tolerância a falhas - *FTU*

O objectivo de uma unidade de tolerância a falhas é mascarar as falhas de um nó. Se um nó implementa uma abstracção *fail-silent*, então a duplicação de nós é suficiente para garantir a falha de um único nó. Caso o nó não implemente *fail-silence*, mas possa exibir no CNI, então deve ser utilizada *redundância tripla*.

6.5.1 Nós *fail-silent*

Numa arquitectura controlada por eventos temporais, uma FTU que consiste em dois *fail-silent* nós, produz zero, um ou duas mensagens correctas de resultados. Se não produzir nenhuma mensagem então falhou. Se produzir uma ou duas mensagens, então o sistema está operacional.

6.5.2 Redundância tripla

Caso um nó exiba falhas nos valores no CNI com uma probabilidade que não possa ser tolerada no domínio de uma dada aplicação, então uma FTU tem de consistir em três nós e um votador.
O votador detecta e mascara erros num nó num só passo através da comparação dos 3 resultados independentes.

6.5.3 FTU Bizantinas

Se não existe nenhuma assumpção acerca dos modos de falha de um nó, então quatro nós são necessários para formar uma FTU que consiga tolerar uma falha *Bizantina* ou *Maligna*
De forma a tolerar falhas *bizantinas* de k nós:

- Cada FTU deve consistir em, pelo menos, $3k + 1$ nós.
- Cada nó deve ser ligado a todos os outros nós da FTU através de $k + 1$ caminhos independentes de comunicação.
- De forma a detectar nós *maliciosos*, $k + 1$ voltas de comunicação devem ser executadas entre os nós. Cada nó deve enviar uma mensagem a todos os outros nós.
- Os nós devem ser sincronizados com precisão.

6.5.4 Serviço de Membros

A falha de uma FTU deve ser reportada convenientemente a todas as outras $FTUs$ e com uma baixa latência. Essa é a função de um serviço de membros.

Arquitectura Controlada por Eventos : O silêncio de um nó pode significar a não ocorrência de eventos ou uma falha *fail-silent* de um nó. Tem então de ser implementado um *watch-dog* para resolver o serviço de membros.

Arquitectura Controlada por Tempo : O estado de membro de uma FTU pode ser detectado durante um período de troca de informação.

6.6 Interligação de um nó reparado

Depois de uma falha no nó, este deve executar um exame próprio. Se este teste for completado com sucesso, o nó pode ser imediatamente reintegrado no sistema, dado que se pode assumir a falta como transiente, não levando a destruição permanente ao hardware do nó.

6.6.1 Encontrar um Ponto de Reintegração

Enquanto uma falha pode ocorrer num instante arbitrário, a reintegração de um nó deve ser cuidadosamente planeada.
O problema consiste em encontrar um instante futuro onde o historial de estado (*H-State*) do nó esteja sincronizado com o resto do sistema.
Num sistema distribuído controlado por tempo, um componente opera periodicamente com um período chamado de *ciclo dos componentes*. Imediatamente a seguir à completação de um ciclo de componente, todas as operações atómicas no componente estão completas e o componente deve estar no *ground-state*. Um instante de *ground-state* é um ponto ideal de reintegração dado que o tamanho do *H-State* é mínimo.

6.6.2 Minimização do *H-State*

Numa primeira fase, todas as estruturas de dados devem ser verificadas quanto à presença de algum *H-State*. É uma boa prática de programação enviar a saída do *H-State* da tarefa numa mensagem especial quando uma tarefa com *H-State* é detectada e reler o *H-State* da tarefa quando a tarefa é reactivada. É assim possível englobar todos os *H-States* das tarefas de um nó numa mensagem *H-State* particular a um dado nó.

6.7 Diversidade de Projecto

Erros de software são erros de projecto que têm a sua origem na complexidade não controlada do sistema.
Existem três grandes estratégias principais para limitar o problema de software não fiável:

- Aumentar o conhecimento de software através da introdução de uma estrutura conceptual e simplificando paradigmas de programação.
- Aplicar métodos formais no processo de desenvolvimento de software para que a especificação possa ser expressa numa forma rigorosa.
- Projectar e replicar diversas versões do software (analogamente ao hardware) para que uma falha numa versão possa ser mascarada pelas outras.

Capítulo 7

Comunicações em Tempo Real

7.1 Requisitos de Comunicação em Tempo Real

Este capítulo apresenta os requisitos necessários ao sistema de comunicação para a implementação de um sistema de Tempo Real Distribuído.

7.1.1 Latência do Protocolo

É o intervalo de tempo entre o início da transmissão no CNI do emissor e a entrega da mensagem no CNI do receptor.

Jitter de latência: *Jitter* que é introduzido pelo protocolo de comunicação. Deve ser o mais pequeno possível e conhecido à priori.

Entrega simultânea em Difusão (Multicast): Num sistema TR distribuido, uma mensagem é enviada para vários nós (topologia multicast). Convém que chegue a todos ao mesmo tempo ou o mais próximo possível.

7.1.2 Suporte a "Composability"

O sistema deve permitir que as características dos nós individuais se mantenham quando interligados. Os meios para atingir este fim são:

- Fazer cumprir as obrigações dos clientes: os clientes não devem saturar o servidor com excessos de pedidos. O sistema de comunicação deve exercer um controlo de fluxo para obrigar a que esta permissa seja cumprida, para que o servidor cumpra as suas deadlines.
- Encapsulamento dos nós

7.1.3 Flexibilidade

Os sistemas mudam ao longo do tempo e o sistema de comunicação deve ser tal que aceite estas mudanças, sem que haja necessidade de voltar a testar o sistema.

7.1.4 Detecção de Erros

Erros de Comunicação: se existe um erro numa mensagem, o sitema deve ser capaz de o corrigir sem aumentar o jitter do protocolo. No limite, se não é capaz, dá a mensagem por perdida e avisa todos os nós aos quais a mensagem diz respeito.

Detecção de Erros nos Nós: uma falha num nó deve ser reconhecida rapidamente e reportada a todos os outros nós. Esta é a funcionalidade do seviço membership.

Reconhecimento Extremo a Extremo: é função do sitema de comunicação fazer o reconhecimento (monotorização) do sucesso, ou insucesso, do envio e recepção de mensagens para evitar que dependa apenas dos nós individuais. Se um nó "avariado"envia uma mensagem errada, o sistema de comunicação deve aperceber-se disso.

7.1.5 Estrutura Fìsica

A estrutura física do sistema de comunicação deve assentar na topologia multicast. Ligações ponto-a-ponto de todos para todos os nós assume custos económicos incomportáveis (por princípio, salvo pressuposto da aplicação do sistema).

Topologia de Barramento de dados (BUS) versus topologia em anel: Depende dos meios usados. Por exemplo, um envio multicast "pede"uma topologia BUS, mas se estivermos a usar fibra óptica, é mais fácil de implementar um anel.

Separação física dos nós que formam uma FTU: Uma FTU (Fault Tolerant Unit) deve ser implementada em topologia BUS, de modo a evitar que a falha de uma das SRU's (Smallest Replaceble Unit) cause falha total do sistema.

7.1.6 Controlo Explícito

Back pressure flow control. O emissor é "pressionado"pelo receptor a gerir o fluxo de informação enviada na medida em que tem que esperar por um reconhecimento de "recepção da última mensagem feita com sucesso".

Exemplo: protocolo Positive-Acknowlegment-or-Retransmission (PAR) - o nome diz tudo. A retransmissão é feita um número limitado de vezes, logo o jitter deste protocolo tem um valor máximo, mas não é sempre igual.

7.1.7 Controlo Implícito

É um protocolo temporal. O emissor e o receptor "acordam"para que, em instantes de tempo especificados, o receptor aceitará todas as mensagens. Este pressuposto impõe a implementação de uma base de tempo global.

7.1.8 Trashing

Fenómeno que consiste na redução da capacidade de tratamento da informação quando o fluxo desta aumenta. Semelhante ao fenómeno de engarrafamento em tráfego urbano.

Causas : os mecanismos mais comuns que causam este fenómenos são os seguintes:

- Retransmissões sucessivas em protocolos PAR
- Escalonamento dinâmico dos serviços de sistemas operativos

A maneira de evitar o fenómeno é efectuar uma repressão mais forte sobre os nós emissores para que estes diminuam o rítmo.

7.1.9 Controlo de fluxo em Sistemas TR

Comparando as características dos dois modos de controlo de fluxo, chegamos à conclusão que as características de controlo implícito de fluxo são as mais adequadas a um sistema TR. Há que evitar, a todo o custo, a sobrecarga no receptor.

Interface entre controlo Implícito e Explícito: quando dois subsistemas usam métodos de controlo diferentes. É difícil de fazer sem alterar as propriedades de cada um. È necessário um buffer para o subsistema em modo explícito.

7.2 Protocolos OSI e TR

7.2.1 O modelo OSI

O modelo em camadas do Protocolo OSI, apesar de conceptual, representa de um modo geral a maneira como os sistemas são implementados de facto. Cada camada traduz no fundo uma interface entre as duas camadas contíguas utilizando um protocolo baseado em PAR. Isto significa que ao longo das (no máximo) sete camadas o jitter de latência será grande e a eficiência reduzida. Deste modo, muitas implementações do protocolo não exercem restrições ao jitter (porque não podem); assim, este protocolo não é aplicável a sistemas de TR.

7.2.2 Tecnologia ATM e Tempo Real

Asynchronous Transfer Mode (ATM) é uma tecnologia utilizada para transmissões com jitter reduzido. O formato das mensagens é encapsulado em pacotes de tamanho fixo "células", que são enviadas periodicamente. Um protocolo de detecção de erro a executar sobre esta tecnologia introduz um jitter de tamanho bem determinado. Esta tecnologia permite a implementação de sistemas RT distribuídos em redes WAN (WIDE AREA NETWORK).

7.2.3 Arquitectura de comunicação em TR

Os principais elementos de uma rede de campo são os seguintes:

- Rede de Campo (Field Bus): interliga sensores e actuadores do sistema
- Rede de Tempo Real: liga os nós do sistema.
- Backbone Network: interligação do sistema com outros sistemas

Field Bus: rede de campo de conexão entre o sistema computorizado e os sensores e actuadores. Transmissão de mensagens curtas com jitter reduzido. Sincronização de relógios. Requisitos temporais devem ser rejeitados.

Rede de Tempo Real: interligação dos nós que formam o cluster computorizado.
Requisitos:

- Transmissão fiável
- Tolerância a falhas
  - nós replicados
  - canais replicados
- Sincronização de relógios
- Serviço ”membership” para detecção de falhas de nós

Backbone Network: existe para troca de informação com outros sistemas, numa relação temporal não restringida.

7.3 Principais conflitos no estabelecimento de protocolos

Existem requisitos das aplicações que nunca deixarão de estar em conflito por nenhuma técnica especial de implementação de protocolos. Apresentam-se alguns casos de seguida.

Service Characteristics	Field Bus	Real-Time Network	Backbone Network
Message semantics	state	state	event
Latency/jitter control	yes	yes	no
Typical data field length	1-6 bytes	6-12 bytes	>100 bytes
Clock synchronization	yes	yes	optional
Fault-tolerance	limited	yes	limited
Membership service	maybe	yes	maybe
Topology	multicast	multicast	point-to-point
Communication control	multi-master	distributed	central or distributed
Flow control	implicit	implicit	explicit
Low Cost	very important	important	not very important

Tabela 7.1: Comparação de field-bus, real-time network e backbone network

7.3.1 Controlo de fluxo externo vs "Composability"

Em caso de controlo externo, se as propriedades temporais do CNI não estiverem bem especificadas, existe a possibilidade de sinais de controlo atravessarem o CNI degradando as características deste nó no que diz respeito a autonomia da rede, ie, o sistema perdeu composability.

7.3.2 Flexibilidade versus Detecção de erros

Um sistema flexível sem replicação para detecção de erros não é possível pois só detecta erros se o comportamento dos nós for conhecido à priori e então deixa de ser flexível

7.3.3 Informação esporádica versus Informação periódica

Dados esporádicos, ou eventos esporádicos, devem ser transmitidos rapidamente. Como não se sabe quando vão suceder, é introduzido um certo grau de incerteza quanto à resposta temporal do sistema.

7.3.4 Unidade única de controlo versus tolerância a falhas

Se o sistema tem uma só unidade de controlo então tem um só ponto sujeito a falha. No entanto, em sistemas com mais controladores mas que usam o sistema de token, se o detentor do token falhar, falha também todo o sistema.

7.3.5 Acesso Probabilístico versus Replicação Determinística

Um sistema que assenta em replicação determinística mas com acesso probabilístico ao meio de comunicação, não garante igualdade das réplicas.

7.4 Protocolos de acesso ao meio físico

São os protocolos que regem o acesso ao meio por parte dos nós. Este capítulo trata apenas protocolos orientados por eventos, já que os orientados pelo tempo são discutidos no capítulo 8.

7.4.1 Características de um canal de comunicação

Largura de banda: É o número limite de bits que podem ser transmitidos por segundos.

Atraso de Propagação: Tempo que a informação demora a percorrer o canal.

7.4.2 CSMA / CD - LON

O protocolo Ethernet é um protocolo CSMA/CD (CD- colision detection) Janelas temporais dedicadas para cada nó

7.4.3 CSMA / CA - (CAN)

Colision avoidance - por um sistema de atribuição de prioridades - os primeiros bits da mensagem.

7.4.4 Token Bus (Profibus)

Uma mensagem circula na rede com um token. Como se viu atrás, o problema reside no atraso não conhecido à priori se o nó que detém o token falha. Tem dois parâmetros:

- Tempo máximo de posse do token
- Tempo máximo de rotação do token (por todos os nós)

7.4.5 Minislotting (ARINC 629)

7.4.6 Central Master (FIP)

Um nó "master"determina quem é que quer ouvir. Funciona por broadcast.

7.4.7 TDMA (TTP)

A capacidade do canal é seccionada e é atribuída uma secção a cada nó. Cada nó transmite na sua secção, ou seja, uma parte da mensagem é de um dado nó.

Capítulo 8

Protocolos de Sistemas activados por Tempo (*TTP*

8.1 Introdução

Protocolos activados por tempo *(TTP)* são projectados para a implementação de um sistema de tempo real duro controlado por tempo. Existem duas versões de TTP, TTP/C para sistemas de tempo real duros com tolerância a falhas, e TTP/A para aplicações de baixo custo

8.1.1 Objectivos do protocolo

Os objectivos do protocolo estão em conformidade com os objectivos delineados no capítulo 7:

- Transporte de mensagens com uma baixa latência e *jitter* mínimo.
- Suporte a *composability*
- Provisão para um serviço de membros tolerante a falhas
- Sincronização de relógio tolerante a falhas
- Controlo distribuído de redundância
- Mínimo peso, quer no tamanho das mensagens, quer no número de mensagens.
- Escabilidade para maiores rácios de dados e uma operação eficiente quer em pares entrelaçados, quer em fibra óptica.

O TTP/C fornece flexibilidade desde que o determinismo possa ser mantido

8.1.2 Estrutura de um sistema TTP

Um TTP é constituído por um cluster de $FTUs$, cada uma sendo constituída por 1,2 ou mais nós interligados por uma rede de comunicação.
Num TTP, um nó é a mais pequena unidade substituível (SRU) que pode ser substituído ou reconfigurado no caso de falhas. O controlador de comunicações num nó tem memória local para guardar o *message descriptor list* ($MEDL$ que determina em que instantes de tempo um nó é autorizado a enviar uma mensagem, e quando pode esperar receber uma mensagem de outro nó.

8.1.3 Regras de Projecto

TTP é um protocolo *time division multiple access* ($TDMA$) onde cada nó envia uma mensagem para o canal partilhado de comunicação durante um intervalo de tempo definido. Como regras de projecto, podem apontar-se:

- *Composability*
- Melhor uso do conhecimento *à-priori*.
- Nomenclatura dos nós.
- Esquemas de confirmação
- *fail-silent* no domínio do tempo
- *fail-silent* no domínio numérico

8.1.4 Variantes do protocolo

Duas variantes do protocolo TTP estão disponíveis; a versão completa TTP/C e a versão reduzida TTP/A. O CNI tem uma estrutura compatível com ambas as variantes.

8.2 Panorâmica dos níveis do protocolo TTP/C

A interface entre o nível de controlo da redundância e a FTU é chamada de *Basic CNI*. A interface entre o nível da FTU e o nível do *host* é chamada de $FTUCNI$

Nível SRU Guarda os campos de dados das mensagens recebidas na memória do CNI de acordo com os dados de controlo de $MEDL$.

Nível de controlo de redundância Gerencia os mecanismos para o *arranque a frio* do cluster TTP/C.

Nível FTU Agrupa dois ou mais nós em $FTUs$. Assegura que dados apenas são visíveis no CNI da FTU depois de ser tornarem permanentes.

8.3 O CNI básico

O CNI é a mais importante interface numa arquitectura accionada por tempo, dado que é a única interface do sistema de comunicação que é visível ao software do *host computer*. Constitui assim a interface programável de uma rede accionada por tempo.

8.3.1 Estrutura do CNI

O CNI básico é uma interface de partilha de dados entre os nível RM e FTU. A sua estrutura é composta principalmente por campos de dados:

Área de Estado/Controlo É uma área do $DPRAM$ que contém informações do sistema. Possibilita a comunicação entre o controloador TTP e o computador principal.

Área de mensagens Contém as mensagens enviadas ou recebidas pelo nó e um *byte* de controlo para cada mensagem. A sua estrutura específica é determinada pelo $MEDL$ do controlador TTP.