UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

Departamento de Informática

Análise do tempo de resposta da composição de micro-protocolos

João Carlos Negrão Ventura

(Licenciado)

Dissertação para obtenção do grau de

Mestre em Informática

Maio de 2001

Análise do tempo de resposta da composição de micro-protocolos

João Carlos Negrão Ventura

Dissertação submetida para provas

de mestrado em

Informática

Departamento de Informática

Faculdade de Ciências da Universidade de Lisboa

Lisboa

Maio de 2001

Trabalho apoiado financeiramente pelo Programa PRAXIS XXI

através da bolsa PRAXIS XXI/BM/20729/99

e pelo projecto DEAR-COTS - PRAXIS/P/EEI/14187/1998

Dissertação realizada sob a orientação do

Doutor Luís Eduardo Teixeira Rodrigues

Professor Auxiliar
Faculdade de Ciências da Universidade de Lisboa

Resumo

Com o aumento do poder de processamento e da largura de banda nas redes de computadores, é possível construir sistemas distribuídos de tempo-real estrito bastante sofisticados. A construção desses sistemas de comunicação usando a composição de vários objectos de micro-protocolos é uma aproximação que tem sido aplicada com sucesso na área de sistemas que não os de tempo-real. Esta estratégia encoraja a reutilização dos componentes dos protocolos e permite que as aplicações configurem pilhas ajustadas às suas necessidades. Para beneficiar desta aproximação em sistemas de tempo-real, é necessário determinar o comportamento temporal de uma composição de protocolos.

Esta dissertação apresenta uma metodologia genérica de análise ao comportamento temporal de pilhas de protocolos derivadas da composição de protocolos. Micro-protocolos individuais são descritos como objectos que recebem e geram eventos; interacções entre protocolos adjacentes são modeladas através da troca desses eventos. A funcionalidade do protocolo é modelada como um conjunto de tarefas, cada uma programada para tratar um evento específico do protocolo.

Para ilustrar o uso da metodologia, é apresentado um estudo da análise temporal de um conjunto de protocolos modulares e tolerantes a falhas desenvolvido para o barramento CAN: o RELCAN e o EDCAN. De modo a realizar este estudo, uma ferramenta informática foi adaptada de forma a satisfazer os requisitos do modelo.

Abstract

With the increase of processing power and network bandwidth it is possible to build sophisticated distributed hard-real time systems. The construction of such communication systems using the composition of several micro-protocol objects is an approach that has been applied with success in the non real-time arena. This encourages the re-use of protocol components and allows the applications to configure stacks tailored to their needs. To benefit from this approach in hard real-time systems, one must be able to derive the timing behavior of a protocol composition given a description of its protocol objects.

This thesis presents a general framework to analyse the timing behavior of protocol stacks derived from the composition of micro-protocols. Individual micro-protocols are described as protocol objects that subscribe and produce events; interactions among adjacent protocols are modeled by the exchange of these events. The protocol implementation is modeled by a set of tasks, each programmed to handle a specific protocol event.

To illustrate the use of the framework, a study is presented on the timing analysis of a set of modular fault-tolerant group communication protocols designed for the CAN field-bus: RELCAN and EDCAN. In order to perform this study, an existing software tool was extended to comply with the model requirements.

Palavras Chave

Tempo-Real

Protocolos de Comunicação

Micro-Protocolos

Análise de Escalonabilidade

Sistemas Distribuídos

Keywords

Real-Time

Communication Protocols

Micro-Protocols

Schedulability Analysis

Distributed Systems

Agradecimentos

Em primeiro lugar desejo agradecer ao Prof. Luís Rodrigues, sem a sua inspirada orientação, motivação e ajuda, este trabalho nunca seria o que é.

A todos os que reviram uma versão resumida deste trabalho, para divulgação na conferência ISORC2001, um obrigado pelos comentários construtivos fornecidos. Uma palavra de especial gratidão ao Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa pelos meios materiais postos à minha disposição e à Fundação para a Ciência e Tecnologia do Ministério da Ciência e Tecnologia pelo apoio financeiro fornecido sem os quais esta dissertação dificilmente seria feita.

A todos os meus colegas da Skysoft Portugal pelo tempo concedido no inicio deste trabalho. Aos meus pais, Carlos e Amélia Ventura, e à minha irmã, Prof. Paula Ventura Martins, pelo carinho e ajuda dispensados. À minha cara-metade, Alexandra Ribeiro e à sua família por me deixarem livre o espaço necessário para poder trabalhar.

Lisboa, Maio de 2001

João Carlos Negrão Ventura

Para a Xana

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nos últimos anos tem-se assistido a uma explosão tanto dos sistemas de tempo-real como dos sistemas distribuídos. Tradicionalmente uma área baseada em sistemas centralizados, em que um único controlador era responsável por um conjunto de sensores e de actuadores, os sistemas de tempo-real começam também hoje em dia a evoluir para os sistemas distribuídos. As motivações para este facto são várias, passando pela crescente complexidade dos sistemas, a sua cada vez maior dispersão espacial, até aos requisitos de tolerância a faltas que levam à replicação dos sistemas críticos. De referir também a criação de novos segmentos de aplicações que exigem resultados em tempo-real e que são por natureza sistemas distribuídos, cujo exemplo mais notório são as aplicações multimédia.

No entanto, estas novas características dos sistemas de tempo-real levantam alguns novos problemas, no que diz respeito à verificação dos tempos de resposta dos mesmos que devem ser sempre inferiores às metas impostas. Estes problemas têm sido abordados de diferentes formas, primeiro em sistemas isolados [19, 14, 18, 32, 3, 2] e depois já em sistemas distribuídos [28, 38, 21], onde além de se considerar o pior tempo de resposta das tarefas que fazem parte do sistema, se entra em conta com o pior tempo de transmissão das mensagens entre os vários computadores do sistema.

Quando se efectua a transmissão de mensagens entre vários computadores, através de uma rede de comunicações, são geralmente usados protocolos de comunicação que têm como principal função o estabelecimento de um canal lógico entre o(s) emissor(es) e o(s) receptor(es). Exemplos bastante conhecidos de protocolos de comunicação são o sistema Open Systems Interconnection (OSI) [12] da International Standards Organization (ISO) e o TCP/IP (Transmission Control Protocol/Internet Protocol) [24] usado na Internet. A utilização deste género de protocolos em ambientes de tempo-real é problemática, visto que a determinação do pior tempo de resposta não é nada simples. Isto leva a que, na maioria das vezes, se usem apenas os serviços básicos destes protocolos de forma a eliminar as características desnecessárias, tentando assim obter um protocolo mais determinista em termos temporais.

Recentemente, com a importação das ideias subjacentes aos modelos orientados aos objectos, começaram a ser desenvolvidos alguns sistemas em que os vários serviços são concretizados através do uso de camadas modulares de micro-protocolos. Este tipo de sistemas é especialmente interessante para ambientes de tempo-real, pois permite uma melhor análise do pior tempo de resposta, uma vez que os protocolos a estudar são bastante mais simples. Por outro lado, ao permitir a inclusão apenas dos serviços necessários, esta aproximação permite obter uma melhoria do tempo de execução dos protocolos. Além disso, ao separar os diferentes serviços em módulos diferentes é facilitada a sua reutilização em diferentes composições.

Esta dissertação tem como objectivo o desenvolvimento de uma metodologia de análise dos piores tempos de resposta de composições de micro-protocolos, ou análise de escalonabilidade, que tire partido das especificidades dos sistemas de micro-protocolos para que os resultados obtidos sejam mais realistas. A metodologia proposta é baseada numa técnica desenvolvida por Tindell [34] que permite tirar partido das relações causais entre as várias etapas de um protocolo. Uma vez apresentados os passos da metodologia, é apresentado um exemplo baseado numa pequena pilha de protocolos baseados no barramento CAN que fornece a este garantias de difusão fiável: o RELCAN e o EDCAN.

A dissertação está organizada da seguinte forma: o capítulo 2 apresenta uma breve introdução aos sistemas de micro-protocolos, o capítulo 3 apresenta a técnica de análise de escalonabilidade usada, o capítulo 4 descreve a metodologia proposta para aplicar a análise de escalonabilidade a sistemas de micro-protocolos, o capítulo 5 apresenta uma aplicação prática da metodologia ao protocolo RELCAN e o capítulo 6 apresenta as conclusões obtidas e algumas direcções para trabalho futuro. Para concluir, é apresentado um breve manual de utilização da ferramenta utilizada para calcular os valores obtidos.

Capítulo 2

Sistemas de Comunicação em Tempo-Real

Com o aumento dos requisitos e da complexidade exigida aos sistemas de tempo-real, não é mais possível pensar nestes como sistemas isolados recebendo dados dos sensores a que estão conectados, processando os dados e, posteriormente, agindo em alguns actuadores. A necessidade de desenvolver sistemas de tempo-real distribuídos leva a que estes dependam de protocolos para comunicação entre os vários nós.

No entanto, os protocolos de comunicação são, em geral, sistemas relativamente complexos que fornecem uma multiplicidade de serviços, mesmo que os sistemas apenas necessitem de alguns deles. Isto acarreta custos no desempenho do sistema, tornando também mais difícil a análise formal do próprio sistema. De modo a minorar estes problemas, alguns grupos começaram a investigar e a desenvolver o conceito de *micro-protocolos*, em que cada serviço é concretizado de forma modular. Através da composição destes micro-protocolos, é possível obter múltiplas configurações diferentes a partir de um pequeno número de módulos base, adaptando os serviços disponibilizados aos requisitos dos sistemas.

2.1 Sistemas de Tempo-Real Distribuídos

Nas últimas décadas, tem-se tornado evidente que cada vez é mais baixo o valor a pagar por um processador com um dado desempenho, e por outro lado, cada vez se desenvolvem problemas mais complexos para serem tratados computacionalmente. É comum hoje em dia dispor-se num vulgar escritório ou laboratório de ensino de um potencial de processamento que poucos anos antes seria incomportável excepto em sistemas paralelos de elevado desempenho (e custo). A possibilidade de aproveitar o poder de processamento de cada uma de várias máquinas, através da sua interligação em rede, de modo a que estas funcionem para os utilizadores como uma única, muito mais potente, naquilo a que se chama um *sistema distribuído*, tem sido alvo de vários projectos de investigação. No entanto, a maioria destes sistemas não são desenhados de forma a satisfazer as especificidades dos ambientes de tempo-real: a necessidade de tomar em conta as metas das tarefas nas decisões de escalonamento, a hierarquização das tarefas em diferentes níveis de prioridade, a necessidade dos serviços serem temporalmente determinísticos, etc. Isto é especialmente relevante em sistemas de tempo-real estrito em que é obrigatório que nenhuma tarefa ultrapasse a meta definida.

Na definição de arquitecturas para sistemas distribuídos em ambiente de tempo-real, existem duas alternativas fundamentais acerca do tipo de estímulos que o sistema recebe: por um lado os sistemas estimulados pelo tempo, e por outro os sistemas estimulados por eventos. A diferença entre eles é que num sistema estimulado pelo tempo as tarefas são activadas em instantes pré-definidos e em sistemas estimulados por eventos as activações ocorrem em reacção à recepção de um qualquer evento. Como exemplos paradigmáticos destas aproximações são apresentados umas descrições breves dos sistemas MARS e XPA[1].

[1]É aconselhada a consulta de [42] para uma introdução mais detalhada

2.1.1 MARS

O sistema MARS (MAintainable Real-time System) [16] é um sistema que tomou como opção básica ser estimulado pelo tempo. Isto significa que toda a recepção de eventos externos apenas acontece em determinados instantes pré-definidos com uma pequena variação devido a diferenças de sincronização entre os diferentes relógios do sistema. Os benefícios desta opção são: uma mais fácil análise formal das garantias de escalonabilidade do sistema e uma mais fácil replicação das tarefas. Por outro lado, este tipo de sistema é menos versátil em situações de carga elevada e no tratamento de eventos esporádicos.

A arquitectura do MARS é composta por agregados (*clusters*), que são compostos por FTUs (Fault Tolerance Units), que por sua vez são compostas por componentes. Cada componente é composto por uma ou mais tarefas, que são as unidades de execução dos programas a executar no sistema. Dentro de uma FTU existem vários componentes que fornecem o mesmo serviço de forma replicada, sendo que alguns destes componentes estão activos e outros inactivos. Devido ao carácter síncrono do sistema, a tarefa de manter todas as réplicas sincronizadas é simplificada. Quando um dos componentes falha, uma das réplicas é activada, passando imediatamente, e de modo transparente para o resto do sistema, a fornecer o serviço. O conjunto de todos os serviços fornecidos pelos FTUs forma o agregado que efectua a funcionalidade pretendida. Além destas funcionalidades, o MARS fornece também um mecanismo de mudança de configuração em tempo de execução que permite a substituição de um conjunto de serviços por outro para melhor lidar com os diferentes modos de funcionamento do sistema. Para mais informações acerca do MARS e dos protocolos activados pelo tempo, consultar [15].

Neste tipo de sistemas, a análise de escalonabilidade é trivial, visto que cada tarefa executa no seu intervalo definido, bastando garantir que o pior tempo de computação seja inferior a este intervalo.

2.1.2 XPA

A arquitectura XPA (eXtra Performance Architecture) [41], por outro lado, é baseada em activação por eventos. Neste tipo de sistema, as acções são causadas pela recepção de um evento. Embora dificulte a análise formal do sistema, esta opção torna o sistema mais adaptável a condições de carga elevada. Um dos objectivos do XPA é ser capaz de fornecer garantias de tempo-real estrito em situações de pouca carga, com uma degradação para um serviço o melhor possível quando a carga for elevada. A arquitectura é também baseada em sistemas de grupos de processos, com vários protocolos de comunicação em grupo que oferecem vários serviços desde a difusão atómica com ordenação total até a mera transmissão de datagramas.

O sistema XPA define um relógio global, sincronizado através de um servidor de tempo distribuído que tira partido das possibilidades de difusão dentro de uma rede local, sendo a variação minimizada através da utilização das garantias dadas por um sistema de tempo-real. O modelo de tolerância a faltas é baseado em replicação havendo suporte para replicação passiva, activa e semi-activa. A técnica mais usada é no entanto a semi-activa, através do uso de um modelo líder-seguidor. Neste modelo, o líder notifica periodicamente os seus seguidores das decisões tomadas de modo a que estes se mantenham sincronizados, havendo a promoção de um seguidor a líder no caso deste falhar.

É para este tipo de sistema que a metodologia de análise do tempo de resposta micro-protocolos proposta nesta dissertação é adequada. Isto porque não é fácil saber em que instantes acontecem os eventos, sendo portanto muito complexa a sua análise de escalonabilidade.

2.2 Protocolos de Comunicação

Para dois sistemas comunicarem entre si através de um canal, é necessária a existência de um conjunto de regras que defina o formato e a sequência que essa comunicação tem de seguir. A este conjunto de regras, é dado o nome de *protocolo de comunicação*, do qual são exemplos bem conhecidos os protocolos definidos pela arquitectura OSI [12] e o TCP/IP [24] usado na Internet.

No área de tempo-real, as arquitecturas de sistemas distribuídos definem os seus próprios protocolos de modo a que estes satisfaçam os requisitos criados pelos serviços existentes na arquitectura. O problema com esta aproximação, é que estes protocolos são difíceis de adaptar de forma a fornecer novas funcionalidades, e de remover parcialmente as que são fornecidas e que não são necessárias. Além disso, a utilização de um protocolo complexo torna complicada a tarefa de analisar o seu pior tempo de execução, levando normalmente a que se faça uma análise demasiado pessimista do sistema.

2.2.1 Micro-protocolos

Um dos sistemas pioneiros em composição de micro-protocolos foi o sistema x-Kernel [11], que inspirou vários outros sistemas como o Horus [40], o Ensemble [9] (tendo estes como antecessor também o ISIS [5]), o Coyote [4], etc. De modo grosseiro, as principais diferenças são o tipo de composições permitidas, sendo fundamentalmente verticais nos sistemas Horus e Ensemble e possivelmente horizontais em sistemas como o Coyote.

Um micro-protocolo é definido como sendo um módulo de código que fornece apenas uma dada propriedade ou funcionalidade de forma independente.

A estrutura típica de um micro-protocolo consiste numa secção de iniciação em que o micro-protocolo inicia as estruturas de dados internas, e vários gestores de eventos, contendo cada um o código que é executado quando o evento respectivo é recebido.

2.2.2 Composição de micro-protocolos

Por definição, um micro-protocolo isolado é funcionalmente pobre, sendo apenas verdadeiramente útil quando integrado com outros numa composição. A maneira como esta composição é feita é uma das características principais dos sistemas de composição de micro-protocolos, podendo ser feita de forma hierárquica, em que cada micro-protocolo se relaciona com os outros através de uma interface e em que os eventos percorrem a pilha sequencialmente, ou numa organização mais livre em que cada micro-protocolo se regista para receber um conjunto de eventos, sendo o processamento desses eventos feito em paralelo por cada micro-protocolo.

Uma das funcionalidades disponibilizadas pelos sistemas de micro-protocolos é assim o registo por parte de cada micro-protocolo dos eventos que está interessado em processar, e dos eventos gerados. O sistema pode então tomar decisões acerca do caminho percorrido por um evento ao longo da pilha e a que micro-protocolos ele deve ser entregue e por que ordem. O sistema tem de ser também capaz de garantir a conformidade de toda a pilha, que deve conter todos os micro-protocolos necessários para um correcto funcionamento desta.

2.3 Sistemas de Micro-Protocolos para Tempo-Real

A aplicação de sistemas de micro-protocolos em sistemas de tempo-real levanta alguns problemas, visto que é necessário garantir de alguma forma que as prioridades dos eventos são respeitadas sem que se originem situações de inversão de prioridade. Este problema e alguns outros foram abordados pelos três sistemas descritos em seguida.

2.3.1 CORDS Paths

O sistema CORDS [39], expande o sistema x-Kernel com a noção de caminhos (*paths*). Estes são uma construção lógica dentro de uma pilha de protocolos do x-Kernel em que os recursos do sistema necessários ao processamento do canal são reservados para uso exclusivo deste.

Os recursos reservados dentro de um caminho são do seguinte tipo:

- Memória: Várias políticas de reserva de memória são definidas desde a pré-alocação na reserva, ou uma simples reserva sem pré-alocação, até à alocação apenas na altura requerida. A arquitectura de reserva de memória é dividida em dois níveis, o exterior que é usado pelos protocolos e o interior que serve de intermediário entre o nível exterior e as várias áreas de memória disponível em exclusivo para cada tipo diferente de alocação interior.

- Processador: O sistema CORDS dispõe de múltiplos fios de execução, sendo que cada mensagem é processada ao longo da pilha de protocolos apenas por um fio. Cada protocolo pode definir quantos fios existem para cada canal, definindo assim a capacidade de processamento concorrente das várias mensagens dentro de um canal.

- Rede: Cada canal deve anunciar o seu padrão de tráfego, de modo a que o sistema possa decidir, na análise de todas os caminhos no seu conjunto, se a rede de comunicações subjacente pode aguentar a carga desejada ou não.

Através desta reserva de recursos, o percurso de um evento através da pilha de micro-protocolos é assegurado de modo a que as suas metas sejam cumpridas.

2.3.2 Canais RTD do CactusRT

Os canais RTD [10] na versão de tempo-real do Cactus, estão assentes sobre o CORDS Paths, sendo este usado para fornecer garantias de tempo-real a um sistema de composição de micro-protocolos. Os micro-protocolos usados no Cactus, similares aos do sistema Coyote [4], são gestores de eventos que concretizam de forma independente um dado serviço, e que através de composição exportam para o exterior uma *interface* compatível com a pilha de protocolos do x-Kernel.

Dentro de cada composição de protocolos no CactusRT, cada evento recebido é processado por um ou mais dos vários micro-protocolos que, independentemente, manipulam os dígitos binários das variáveis de controlo que determinam quando é que um dado evento pode ser entregue à próxima camada (apenas quando todos os micro-protocolos sinalizarem a sua permissão é que essa acção pode ser realizada).

A estrutura de garantia temporal do CactusRT assenta também sobre alguns serviços, nomeadamente o módulo de controlo de canais (CCM) e o módulo de controlo de admissão (ACM). O CCM fornece uma *interface* para as aplicações solicitarem a criação de canais. O ACM é um serviço distribuído que tem como função a permissão ou recusa de reserva de recursos na criação dos canais. Parte das funções do ACM são a verificação de escalonabilidade local dos micro-protocolos, e a disponibilidade dos recursos pretendidos.

2.3.3 *RT-Appia*

O *RT-Appia* [29], é uma extensão do *Appia* [20] para sistemas de tempo-real. Os sistemas *Appia* estão organizados da seguinte maneira: cada micro-protocolo é uma camada, e a cada instância de uma camada é dado o nome de sessão. As camadas são organizadas de forma ordenada de modo a fornecer uma qualidade de serviço desejada a cuja instanciação se dá o nome de canal. As sessões podem ser partilhadas por vários canais. Cada camada declara três conjuntos de eventos de modo a poder ser aplicado um algoritmo de composição de protocolos, que são o conjunto de eventos gerados, o conjunto de eventos recebidos e o conjunto de eventos requeridos para uma execução correcta. Para melhorar o desempenho de cada canal, a rota de cada evento é definida aquando da criação do canal, permitindo assim que os eventos transitem apenas para as sessões que o processam.

O sistema *Appia* foi desenvolvido em Java, o que o torna facilmente reutilizável em várias arquitecturas e sistemas. O *RT-Appia* pretende usar a versão de tempo-real do Java [6], de modo a expandir o *Appia* para que este forneça garantias de tempo-real.

2.4 Sumário

Neste capítulo foram apresentados algumas arquitecturas especialmente desenvolvidas para utilização em sistemas distribuídos em ambientes de tempo-real. No entanto, os serviços de comunicações fornecidos por estas arquitecturas são específicos da mesma, e geralmente pouco flexíveis. Isto leva a que as aplicações tenham de ser escritas tendo em conta as especificidades de uma dada arquitectura, o que torna complicada a tarefa de a adaptar no futuro a outra arquitectura diferente. Parte da solução deste problema pode passar pela utilização de sistemas de composição de micro-protocolos que tenham sido desenvolvidos para utilização em ambientes de tempo-real. Estes sistemas permitem mais facilmente a reutilização dos protocolos modulares e, através da remoção de serviços indesejados, uma redução do tempo de computação dos próprios protocolos.

Um dos principais requisitos em sistemas de tempo-real estrito é a obrigatoriedade que todas as tarefas cumpram as metas definidas. De modo a assegurar que isto assim acontece, é necessário realizar uma análise temporal do sistema em causa, ou seja uma análise de escalonabilidade. A separação dos serviços dos micro-protocolos em módulos não facilita por si esta tarefa, tornando-a até mais complicada devido ao aumento do número de componentes do sistema a analisar. No entanto, desde que se saiba como aproveitar as peculiaridades destes sistemas, é permitida uma melhor aplicação da análise, reduzindo até algum do pessimismo associado. No próximo capítulo será apresentada uma técnica para realizar análises de escalonabilidade, que será depois usada como base para aplicação a sistemas de micro-protocolos.

Capítulo 3

Análise de Escalonabilidade

De modo a poder utilizar os sistemas de composição de micro-protocolos em ambientes de tempo-real estrito[1], é preciso provar de modo formal que uma dada configuração cumpre sempre as metas. Esta tarefa está longe de ser trivial devido ao enorme número de possibilidades existentes mesmo em sistemas simples. Ao acto de verificar se uma dada configuração das várias tarefas que compõem um sistema, com uma certa ordem de escalonamento, cumpre ou não as metas dá-se o nome de *análise de escalonabilidade*[2].

Neste capítulo é apresentada uma técnica de análise de escalonabilidade que se adequa ao objectivo de aplicação a sistemas de micro-protocolos, pois incorpora a noção de desfasamentos de tempo (*offsets*), que permitem capturar de forma adequada a noção das relações causais de precedência entre as várias tarefas que compõem um sistema deste tipo.

[1]Os sistemas de tempo-real estrito são aqueles em que as metas nunca podem ser ultrapassados, enquanto os sistemas de tempo-real lato são aqueles em que esta condição é relaxada. Exemplos de sistemas de tipo estrito são os sistemas de controlo de centrais nucleares e de aviação; os sistemas multimédia são exemplos de sistemas de tipo lato

[2]Uma boa introdução ao assunto é dada em [7].

3.1 Análise de Escalonabilidade em um só processador

Uma das primeiras técnicas de Análise de Escalonabilidade foi a *Rate Monotonic Analysis* (RMA) apresentada em [19]. Nesta técnica, as prioridades das tarefas são atribuídas numa proporção inversa à sua taxa de chegada (quanto menor for esta, maior a prioridade). Nestas condições, o sistema é escalonável se cumprir a seguinte condição:

$$\sum_{i=1}^{N} \left(\frac{C_i}{T_i} \right) \leq N(2^{1/N} - 1) \tag{3.1}$$

em que N é o número total de processos no sistema, C_i e T_i o pior tempo de computação e a taxa de chegada do processo i, respectivamente. A primeira parte desta equação calcula a utilização total do sistema ($U = C/T$), sendo que para cada N existe um valor que a majora. Para N grandes, este majorante aproxima-se assimptóticamente de $\log_e 2 \simeq 69,3\%$. Este foi o primeiro grande resultado em análise de escalonabilidade: um sistema cuja utilização máxima de todos os processos seja inferior a 69,3% é sempre escalonável desde que as prioridades sejam atribuídas de forma inversamente proporcional à taxa de chegada.

3.1.1 Método simples

No entanto, a análise RMA não oferece garantias de escalonabilidade para sistemas em que a condição 3.1 não seja cumprida. Para estes sistemas é necessário o uso de outras técnicas de análise. O método simples de análise apresentado a seguir foi derivado por Joseph e Pandya [14], e usado por Tindell em [33] e [37]. Este método pressupõe que as tarefas são esporádicas e com um tempo mínimo entre ocorrências. Todas as técnicas discutidas daqui em diante funcionam através do cálculo dos piores tempos de resposta das tarefas. O teste de escalonabilidade passa pois a ser uma comparação entre estes valores e as metas definidas.

O pior tempo de resposta r_i de uma tarefa i é dado pela seguinte equação[3]:

$$r_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i}{T_j} \right\rceil C_j \tag{3.2}$$

sendo C_i o pior tempo de computação da tarefa i, $hp(i)$ o conjunto das tarefas de maior prioridade do que i, e T_j o tempo de intervalo médio entre invocações sucessivas da tarefa j (no caso de j ser periódica, T_j é igual ao período). Esta equação calcula o pior tempo de resposta de uma tarefa a partir das contribuições do pior tempo de computação da própria tarefa, e dos piores tempos de computação das outras tarefas de maior prioridade que podem ser accionadas várias vezes durante a execução desta.

A ocorrência de r_i em ambos os lados da equação 3.2 faz com que a sua determinação não seja trivial, sendo necessário fazê-lo por iterações sucessivas através da seguinte equação modificada:

$$r_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i^n}{T_j} \right\rceil C_j \tag{3.3}$$

Se a utilização ($U = C/T$) dos i níveis de prioridade mais elevados for menor do que 1, esta equação converge garantidamente (isto porque as sucessivas aproximações a r_i são monotonamente crescentes, para mais detalhes consultar [14] ou [33]). As iterações começam com $r_i^0 = C_i$ como valor inicial.

3.1.2 Tarefas com Metas Arbitrárias

A equação 3.2 é suficiente para o caso em que as tarefas não ultrapassam as suas metas. Quando isto não acontece, é necessário entrar em conta com as instâncias anteriores ao

[3]A notação $\lceil x \rceil$ indica o primeiro inteiro maior ou igual a x.

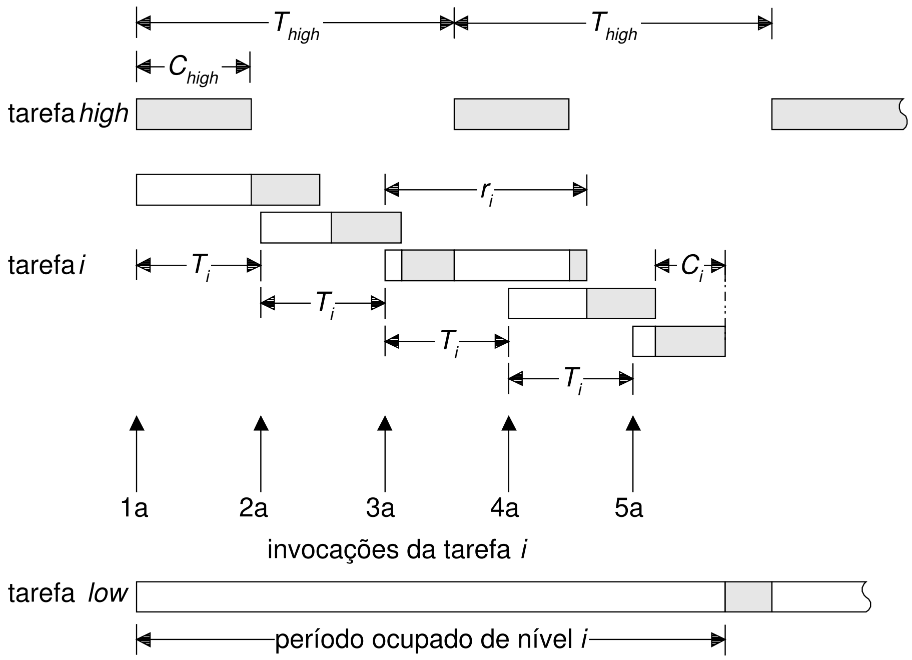

Figura 3.1: Período ocupado de nível i

determinar r_i. De modo a tratar este problema, usa-se a noção de "períodos ocupados": um "período ocupado de nível i" é definido como o tempo máximo em que um processador executa tarefas de nível de prioridade igual ou superior à prioridade da tarefa i. Este conceito foi introduzido por Lehoczky em [17] para tratar este tipo de casos.

Durante um período ocupado de nível i, são executadas várias instâncias anteriores da tarefa i, que são ocasionalmente suspensas para permitir a execução de tarefas de maior prioridade. É então necessário descobrir qual o tamanho deste período, considerando as instâncias que começaram nos instantes T_i, $2T_i$, $3T_i$, $\ldots$, qT_i antes do instante de activação da invocação actual.

Um período ocupado de nível i que comece no instante qT_i antes da invocação actual da tarefa i, tem como tamanho:

$$w_i(q) = (q+1)C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i(q)}{T_j} \right\rceil C_j \tag{3.4}$$

onde o termo qC_i significa a contribuição das q instâncias da tarefa i que começaram a executar no instante 0 e que ainda podem estar activas. Para resolver esta equação é também preciso iterar $w_i(q)$, análogamente ao usado na equação 3.3, usando um valor inicial de $w_i(0)^0 = C_i$. A figura 3.1 ilustra de forma clara a contribuição das tarefas de maior nível de prioridade para o período ocupado.

Uma vez obtido o tamanho do período ocupado, é possível extrair o pior tempo de resposta da tarefa i. Este é o máximo dos valores obtidos ao subtrair do tamanho do período ocupado o tempo qT_i, de forma a obter apenas o tempo de resposta da invocação actual da tarefa i:

$$r_i = \max_{q=0,1,2,\ldots} (w_i(q) - qT_i) \tag{3.5}$$

É necessário verificar valores de q cada vez maiores, até que se encontre um "período ocupado de nível i" que acabe antes da invocação actual da tarefa i, pois nessa altura já não existe mais nenhuma contribuição para r_i de tarefas de nível de prioridade igual ou superior a i. Para isso, apenas é necessário considerar valores de q tal que $w_i(q) < (q+1)T_i$, parando quando esta condição já não se verificar.

É possível ver que, no caso em que $q = 0$, a equação 3.5 é equivalente à equação 3.2.

3.1.3 Extensão para o Protocolo de Tecto da Prioridade

Em sistemas multi-tarefa com escalonamento por preempção e mecanismos de controlo de acesso a estruturas partilhadas, é possível que uma tarefa fique bloqueada por uma outra tarefa de menor prioridade que detenha o direito de acesso a uma estrutura partilhada. Este comportamento não é inesperado, sendo até o que se pretende ao usar mecanismos de controlo de acesso. Entretanto, uma outra tarefa com prioridade intermédia interrompe a tarefa de menor prioridade e passa a executar, continuando a tarefa de maior prioridade bloqueada. A este problema dá-se o nome de *Inversão de Prioridade*, que quando não é devidamente acautelado pode dar origem a cenários de bloqueio circular por parte de várias tarefas[4].

O problema da inversão de prioridade foi resolvido por Sha, Rajkumar e Lehoczky em [32] através do uso de protocolos de herança de prioridade, e onde foram estudadas as propriedades de um protocolo desta família chamado Protocolo de Tecto da Prioridade (*Priority Ceiling Protocol*).

Este protocolo funciona da seguinte forma: uma tarefa que solicite o acesso exclusivo a uma estrutura partilhada apenas vê o seu pedido satisfeito caso a sua prioridade dinâmica seja maior que o tecto de prioridade de todas as outras estruturas que estejam atribuídas a outras tarefas. A prioridade dinâmica é igual à sua prioridade estática a não ser que a tarefa esteja na secção crítica e bloqueie tarefas de maior prioridade, caso em que herda a prioridade mais elevada das tarefas bloqueadas. Ao sair da zona crítica, a tarefa passa a executar com a sua prioridade estática.

Para incluir a contribuição do Protocolo de Tecto da Prioridade, inclui-se o factor B_i na equação 3.4:

$$w_i(q) = (q+1)C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i(q)}{T_j} \right\rceil C_j \tag{3.6}$$

A determinação do factor B_i pode ser feita de acordo com o definido em [32]. De modo sucinto, B_i é a soma do pior tempo de computação de qualquer secção crítica

[4]Por exemplo, a sonda *Pathfinder* da NASA que chegou a Marte em 4 de Julho de 1997, começou a sofrer umas reiniciações inesperadas alguns dias depois do início da missão, devido a uma inversão de prioridade entre uma tarefa A de gestão do barramento de dados (alta prioridade), uma tarefa B de recolha de dados meteorológicos (baixa prioridade) e uma tarefa M de comunicações (média prioridade). Normalmente, B colocava os dados no barramento, adquirindo um acesso exclusivo de acesso a este , podendo ser interrompida por A que ficaria depois bloqueada até que B libertasse o acesso exclusivo. O problema levantava-se quando M interrompia depois B, impedindo esta de executar durante um largo período de tempo, durante o qual um temporizador de alarme era activado devido à não execução de A, accionando uma reiniciação total do sistema. A resolução do problema foi feita com uma actualização remota do sistema a partir da Terra.

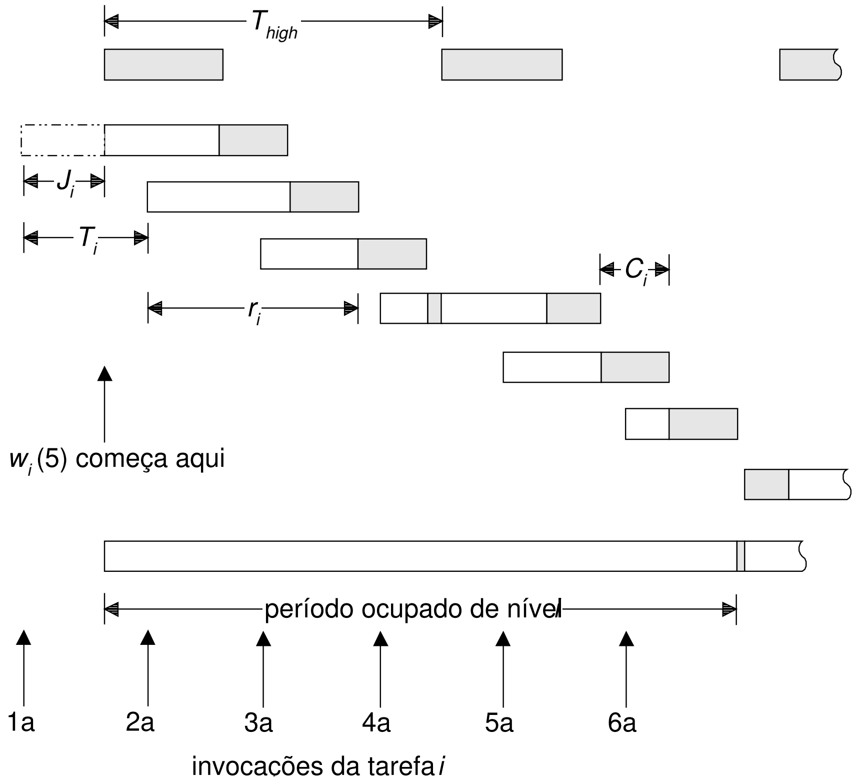

Figura 3.2: Efeitos do *jitter* no período ocupado de nível i

das tarefas de menor prioridade que podem bloquear i, por cada vez que i possa ser suspendido ao pedir acesso a uma estrutura partilhada.

3.1.4 Efeitos do *jitter*

As equações anteriores supõem que a tarefa é colocada na fila de execução no instante em que teoricamente é activada. Infelizmente isto nunca acontece, devido à incerteza inerente ao funcionamento dos computadores. A este efeito, dá-se o nome de *jitter*.

De forma a estender as equações anteriores para incluir o *jitter*, é necessário incluir o termo J_i nas equações 3.5 e 3.6:

$$r_i = \max_{q=0,1,2,\ldots} (w_i(q) + J_i - qT_i) \tag{3.7}$$

$$w_i(q) = (q+1)C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{J_j + w_i(q)}{T_j} \right\rceil C_j \quad (3.8)$$

O efeito do *jitter* no período ocupado é facilmente visível na Fig. 3.2, a primeira invocação da tarefa i, que deveria começar a executar antes da tarefa de maior prioridade, é retardada, adiando também o início do período ocupado, sendo pois necessário somar o valor do *jitter* ao tamanho deste período nas expressões das 3.5 e 3.6 onde aparece o termo $w_i(q)$.

3.1.5 Efeitos do Escalonador

Os sistemas operativos multi-tarefa têm habitualmente um escalonador do tipo *tick scheduler*, que é periodicamente executado após um certo número de batidas do relógio (geralmente na ordem dos micro-segundos). De modo a incluir o efeito desta execução é necessário fazer algumas alterações nas equações de determinação do pior tempo de computação. Os escalonadores, de cada vez que são activados, demoram habitualmente mais tempo a mover a primeira tarefa da fila de pendentes para a fila de execução do que a moverem as tarefas seguintes (devido aos custos associados a iniciações internas). Para considerar esta diferença, atribui-se a C_{QL} o custo de mover a primeira tarefa, e a C_{QS} o das tarefas seguintes.

É também preciso incluir o tempo de computação do próprio escalonador. Este é modelado com um pior tempo de computação C_{clk} e período T_{clk}. Dentro de um intervalo w, o número máximo de activações do escalonador (L) é então:

$$L(w) = \left\lceil \frac{w}{T_{clk}} \right\rceil \quad (3.9)$$

De cada vez que é activado, o número de tarefas que o escalonador coloca na fila de execução é:

$$K(w) = \sum_{\forall j \in hp(i)} \left\lceil \frac{J_j + w}{T_j} \right\rceil \tag{3.10}$$

No caso em que, no intervalo w, o número de activações do escalonador (L), seja maior que o número de tarefas a executar (K), no pior caso todas as colocações serão ao pior custo (C_{QL}), demorando portanto $LC_{clk} + KC_{QL}$. Se pelo contrário, K for maior que L ao longo de w, então apenas as primeiras L colocações são ao pior custo e as restantes apenas a C_{QS}. O factor $\tau_i(w)$ dado na seguinte equação exprime ambos os casos:

$$\tau_i(w) = L(w)C_{clk} + \min(L(w), K(w))C_{QL} + \max(K(w) - L(w), 0)C_{QS} \tag{3.11}$$

Incluindo este factor na equação 3.8 fica:

$$w_i(q) = (q+1)C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{J_j + w_i(q)}{T_j} \right\rceil C_j + \tau_i(w_i(q)) \tag{3.12}$$

3.1.6 Desfasamentos de Tempo entre tarefas

Em [34], Tindell investigou um importante conceito para sistemas em que as tarefas interagem entre si, os desfasamentos de tempo entre tarefas.

As tarefas estão então organizadas em conjuntos de tarefas denominados transacções. Uma transacção ti tem um período T_{ti}. Uma tarefa i da transacção ti é activada a cada e_i invocações da transacção, tendo portanto um período e_iT_{ti}. Uma tarefa é executada num instante fixo após o início da transacção. Este desfasamento, ou *offset*, é dado por O_i. Tal como antes, existe algum *jitter*, pelo que a tarefa começa efectivamente a executar entre O_i e $O_i + J_i$.

3.1.6.1 O Pior Tempo de Resposta da tarefa i

O tempo de resposta da tarefa i é o intervalo de tempo que começa em O_i e acaba no final da execução de i. Esta computação pode incluir ainda computações de instâncias anteriores de i, fazendo com que apenas acabe no final do período crítico de nível i. Para calcular o pior tempo de resposta de um período ocupado que inclua i, é preciso investigar os períodos ocupados anteriores, tal como na secção 3.1.2. Para isso, percorrem-se todas as anteriores qe_i invocações da transacção ti, com $q = 0, 1, 2, \ldots$. O período crítico começa no tempo W_{ti} após o inicio da transacção, com a execução de uma tarefa j arbitrária, mas que faz parte do conjunto de tarefas da transacção ti, começando a executar $O_j + J_j$ após o inicio da transacção.

$$r_i = \max_{q=0,1,2,\ldots} \left(\max_{\forall j \in tasks(ti)} (w_{i,q} + O_j + J_j - T_{ti}(qe_i + v_{i,ti}) - O_i) \right) \tag{3.13}$$

$$v_{i,ti} = \left\lceil \frac{O_j + J_j - O_i - J_i}{T_{ti}} \right\rceil \tag{3.14}$$

A equação 3.13 dá o pior tempo de resposta da tarefa i, sendo ti a transacção onde i está incluída, $tasks(ti)$ o conjunto das tarefas de ti, $w_{i,q}$ o tamanho do pior período ocupado. O pior tempo de resposta de i conta desde o início da execução até ao fim do período ocupado, pelo que ao tamanho deste acrescido de $O_j + J_j$ é preciso descontar o período de todas as transacções anteriores à actual e o desfasamento de i. A equação 3.14 serve como "função normalizadora" para garantir que o período ocupado começa antes da última activação de i.

3.1.6.2 O Tamanho do Pior Período Ocupado

Para calcular o tamanho do pior período ocupado, é necessário calcular qual a interferência de cada uma das tarefas da transacção ti, o que é dado pelo número de activações

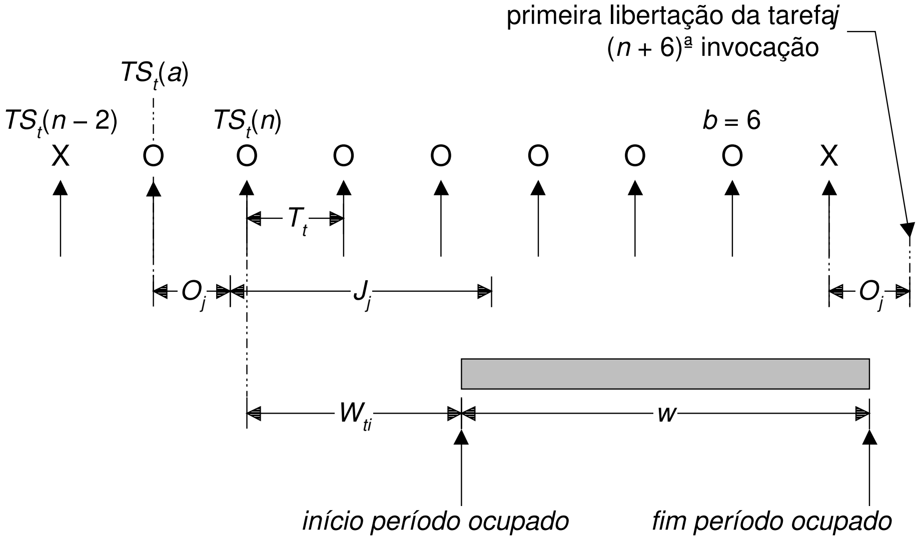

Figura 3.3: A interferência da tarefa j durante o período ocupado w

dessas tarefas dentro do período ocupado a multiplicar pelo pior tempo de computação de cada uma. O número de activações de uma tarefa j dentro de um qualquer período ocupado é dado pelo tempo entre o início da transacção que lhe dá origem e o seu fim $(W_{ti} + w)$ acrescido de um T_{ti} por cada tarefa que devido a um grande $O_j + J_j$ só comece a executar dentro do período crítico $(-v_{j,ti}T_{ti})$ e descontando O_j, tudo a dividir pelo período da tarefa j (e_jT_{ti}) (ver ilustração na Fig. 3.3).

$$I_{i,ti} = \sum_{\forall j \in hp(i) \cap tasks(ti)} \left\lceil \frac{W_{ti} + w - O_j - v_{j,ti}T_{ti}}{e_jT_{ti}} \right\rceil C_j \tag{3.15}$$

Uma vez conhecida a interferência devida às tarefas de ti, é possível calcular a computação total devida a uma tarefa i, somando B_i (o bloqueamento por tarefas de menor prioridade que i), o efeito da actual e das q instâncias anteriores de i ($(q+1)C_i$) e a interferência de todas as tarefas de maior prioridade de todas as transacções. O total de computação devido a i é então:

$$B_i + (q+1)C_i + \sum_{\forall t \in trans} I_{i,t} \tag{3.16}$$

A partir deste resultado é possível obter o valor para $w_{i,q}$, o tamanho do pior período ocupado de nível i, visto que ele é igual ao tempo gasto em computação de tarefas de nível de prioridade i ou superior.

$$w_{i,q} = B_i + (q+1)C_i + \sum_{\forall t \in trans} I_{i,t} \tag{3.17}$$

Uma vez que w ocorre de ambos os lados da equação, é necessário iterar sucessivamente tal como foi feito para a equação 3.3. A determinação exacta da equação 3.17 é infelizmente muito complicada, porque é preciso avaliar todos os possíveis valores de W para todas as transacções:

$$\max_{\substack{\forall j \in tasks(1) \\ W_1 = O_j + J_j}} \left(\max_{\substack{\forall j \in tasks(2) \\ W_2 = O_j + J_j}} \cdots \left(\max_{\substack{\forall j \in tasks(N) \\ W_N = O_j + J_j}} (w) \right) \right) \tag{3.18}$$

Onde N é o número total de transacções. A avaliação desta equação tem de ser feita $n_1 \times n_2 \times n_3 \times \ldots \times n_N$ vezes o que a torna computacionalmente proibitiva.

É possível modificar a equação 3.13 de forma a usar os valores de W obtidos na equação acima:

$$r_i = \max_{q=0,1,2,\ldots} (w_{i,q} + W_{ti} - T_{ti}(qe_i + v_{i,ti}) - O_i) \tag{3.19}$$

3.1.6.3 Análise Tratável

Embora o cálculo exacto da equação 3.17 seja difícil, é possível simplificá-lo bastando para isso encontrar um valor de W_t que maximize a interferência para uma transacção t. Isto é feito através do uso da função maxv[5], que devolve o valor que maximiza o seu argumento.

Durante as iterações sucessivas de $w^n_{i,q}$, modificando a equação 3.17, o valor de W_t

[5] Se $v' = \mathrm{maxv}_{\forall v \in r}(f(v))$, então $\forall v \in r, f(v) \leq f(v')$.

pode então ser:

- O valor $W_{ti} = O_j + J_j$ obtido na equação 3.13, se $t = ti = trans(i)$.
- $W_t = O_k + J_k$, sendo $k \in tasks(t)$ de forma a maximizar a interferência $I_{i,t}$:

$$W_t = \operatorname{maxv}_{\substack{\forall k \in tasks(t) \\ W_t = O_k + J_k}} (I_{i,t}) \tag{3.20}$$

Finalmente, e para terminar a análise tratável, a equação 3.13 define uma sequência infinita de q. Não é necessário considerar um número infinito de períodos ocupados, bastando parar quando a seguinte condição se verificar:

$$\forall k \in tasks(ti), (w_{i,q} + O_k + J_k \leq T_{ti}((q+1)e_i + v_{i,ti}) + O_i + J_i) \tag{3.21}$$

3.1.6.4 Efeitos do Escalonador

Tal como na secção 3.1.5 é possível estender as equações anteriores de forma a incluir o tempo gasto pelo escalonador em mover tarefas para a fila de execução. Como anteriormente, vamos utilizar o factor $\tau_i(w)$ dado pela equação 3.11, em que o factor L continua a ser dado pela equação 3.9, mas em que o factor K passa a ser dado por:

$$K = \sum_{\forall t \in trans, \forall j \in tasks(t)} \left\lceil \frac{W_t + w - O_j - v_{j,t} T_t}{e_j T_t} \right\rceil \tag{3.22}$$

As equações anteriores (3.17 e 3.20) passam então a ser:

$$w_{i,q} = \tau_i(w) + B_i + (q+1)C_i + \sum_{\forall t \in trans} I_{i,t} \tag{3.23}$$

$$W_t = \operatorname{maxv}_{\substack{\forall k \in tasks(t) \\ W_t = O_k + J_k}} (I_{i,t} + \tau_i(w)) \tag{3.24}$$

3.2 Trabalho relacionado

Apesar do modelo apresentado até agora ser já bastante completo, possui ainda algumas limitações. No trabalho desenvolvido por Palencia e Harbour [8, 21, 22], são removidas as limitações do desfasamento ter de ser menor que o período e são introduzidos desfasamentos dinâmicos. Estas extensões ao modelo são conseguidas de forma conceptualmente simples, mas com alguma complexidade nas alterações das fórmulas, pelo que não serão descritas em pormenor. A introdução de desfasamentos maiores que o período é conseguida através do uso de desfasamentos modificados através da seguinte função $\Phi_i = O_i \bmod T_i$, e os desfasamentos dinâmicos são limitados a um intervalo $[O_{i,min}, O_{i,max}]$, sendo aplicados na análise os valores $O_i = O_{i,min}$ e $J_i = J_i + O_{i,max} - O_{i,min}$. Além destas extensões, estes trabalhos incluem algum tratamento de forma a reduzir o pessimismo da análise através do recurso a estudos do melhor caso (alterando o *jitter* de modo a que este corresponda à diferença entre o pior e o melhor caso) e explorando melhor as relações de precedência entre tarefas.

Complementarmente à análise desenvolvida, foi proposto por Audsley em [1] uma técnica de atribuição de prioridades que é considerada óptima. A técnica proposta funciona através da atribuição de diferentes prioridades às várias tarefas a escalonar até que todas sejam escalonáveis ou até à determinação de que o sistema é impossível de escalonar.

3.3 Análise de Escalonabilidade em Sistemas Distribuídos

Em relação aos sistemas de um só processador, os sistemas distribuídos apenas incluem a necessidade de as tarefas ficarem bloqueadas à espera de mensagens vindas de tarefas dos sistemas remotos.

De forma a estender a análise anterior para incluir as comunicações entre tarefas localizadas em processadores ligados em rede, é preciso calcular qual o pior tempo entre o envio e a entrega das mensagens. O modelo usado, desenvolvido em [38], pressupõe que as mensagens são enviadas usando TDMA (*Time Division Multiple Access*) e que cada receptor apenas pode receber um tipo de mensagem (mais do que uma podem ser permitidas usando tarefas específicas de recepção que copiam os dados da mensagem para uma área comum).

Para isto é necessário calcular dois factores, o pior tempo entre o envio da mensagem e a sua chegada ao destino (o pior tempo de chegada), e o seu posterior tempo de entrega à tarefa de destino.

3.3.1 O pior tempo de chegada

Para determinar o pior tempo de chegada, é útil supor que o canal de comunicações é um processador e que as mensagens são tarefas. Deste modo, podemos reutilizar a equação 3.7, alterando os factores:

$$a_m = \max_{q=0,1,2,\ldots} (w_m(q) + X_m(q) - qT_m) \tag{3.25}$$

sendo a_m o pior tempo de chegada da mensagem m, $w_m(q) - qT_m$ o tempo que a mensagem passa na fila de envio (permitindo até q mensagens m atrasadas) e $X_m(q)$ o tempo que a mensagem demora a ser transmitida.

O número de pacotes que estão na fila de envio antes de m (I_m) é dado por:

$$I_m(w) = \sum_{\forall j \in hp(m)} \left\lceil \frac{w + r_{s(j)}}{T_j} \right\rceil P_j \tag{3.26}$$

sendo $hp(m)$ o conjunto de mensagens com maior prioridade do que m, $r_{s(j)}$ é o pior tempo de computação da tarefa emissora da mensagem j, e P_j é o número de pacotes que constituem a mensagem j.

O número de pacotes que têm de ser enviados para que m seja transmitida são:

$$x(q) = (q+1)P_m + I_m(w_m(q))b \tag{3.27}$$

O número de *slots* necessários para os transmitir é:

$$s(q) = \left\lceil \frac{x(q)}{S_p} \right\rceil \tag{3.28}$$

sendo S_p o número de pacotes que o processador p pode transmitir no seu *slot*. Multiplicando este valor por T_{TDMA}, o período do ciclo TDMA, obtemos o tempo dispendido para transmitir todos estes pacotes:

$$w_m(q) = s \times T_{TDMA} \tag{3.29}$$

O tempo que m demora a ser transmitida é então:

$$X_m(q) = \wp + \rho(x - (s-1)S_p) \tag{3.30}$$

onde $\wp$ significa um factor constante de atraso de propagação, ρ é o tempo de transmissão de um pacote na rede.

3.3.2 O pior tempo de entrega

Após a mensagem m chegar ao tampão (*buffer*) de recepção do computador remoto, ainda decorre algum tempo até que o gestor de pacotes a entregue à tarefa de destino.

De forma similar à equação 3.26, podemos construir a seguinte equação que indica o máximo número de vezes que o *packet handler* é activado:

$$l_p(w) = \sum_{\forall k \in recebidas(p)} \left\lceil \frac{w + r_{s(k)} + a_k + J_{entrega}}{T_k} \right\rceil P_k \tag{3.31}$$

onde $recebidas(p)$ é o conjunto das mensagens destinadas ao processador p, $r_{s(k)}$ é o pior tempo de resposta da tarefa emissora de k, a_k é o pior tempo de chegada de k, e T_k é o período da mensagem k.

Na equação 3.12, o termo $(q+1)C_i$ significa o pior tempo de computação devido à tarefa i; ao calcular o pior tempo de resposta da tarefa $entrega$, podemos substitui-lo por:

$$u_{entrega,q}(w) = \min(l_p(w), q+1)C_{entrega} \tag{3.32}$$

Ficando então, com $i = entrega$:

$$w_i(q) = u_{i,q}(w_i(q)) + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{J_j + w_i(q)}{T_j} \right\rceil C_j + \tau_i(w_i(q)) \tag{3.33}$$

O tempo de entrega, $r_{entrega}$ é pois (da equação 3.7):

$$r_{entrega} = \max_{q=0,1,2,\ldots} (w_{entrega}(q) + J_{entrega} - qT_{entrega}) \tag{3.34}$$

3.3.3 O pior tempo entre o envio e a entrega

Juntando os resultados anteriores, o pior tempo entre o envio e a entrega da mensagem m é:

$$r_m = a_m + r_{entrega} \tag{3.35}$$

que é uma simples soma do pior tempo de chegada com o pior tempo de entrega. Isto conclui o modelo de rede genérico usando acesso TDMA.

3.4 Sumário

Neste capítulo foi apresentada uma técnica de análise de escalonabilidade desenvolvida por Ken Tindell, que partindo de uma técnica simples, cresce em complexidade até ser capaz de tomar em conta com os efeitos de invocações anteriores da tarefa, o efeito do bloqueio devido a tarefas de menor prioridade (usando protocolos de herança de prioridade), o *jitter* esperado no sistema, os efeitos do escalonador e especialmente, a noção de desfasamentos de tempo entre as tarefas. A possibilidade de utilização de desfasamentos permitirá à metodologia apresentada no próximo capítulo capturar a noção de causalidade no tempo entre as tarefas que realizam os micro-protocolos, separando a análise em dois modelos distintos: o modelo da computação nos vários nós de um sistema distribuído, e o modelo de transmissão de mensagens na rede de comunicação. É também apresentado um modelo de rede, que prevê a fragmentação da mensagem em vários pacotes e o tempo dispendido por estes nas filas de envio e recepção, embora sem noções de prioridade e com acesso TDMA.

Capítulo 4

Análise de Micro-Protocolos para Tempo-Real

Originalmente desenvolvida para sistemas embebidos, a análise de escalonabilidade não é fácil de aplicar a sistemas distribuídos, especialmente a sistemas com tanta interconectividade como os protocolos de comunicação. Neste capítulo, será apresentada uma metodologia que facilita a aplicação da análise de escalonabilidade a pilhas obtidas por composição de micro-protocolos.

Durante a apresentação da metodologia, esta será ilustrada com um pequeno exemplo, mas apenas nos parâmetros que não são específicos aos ambientes reais (estes serão ilustrados com um exemplo mais completo no cap. 5). O pequeno exemplo consiste nu-

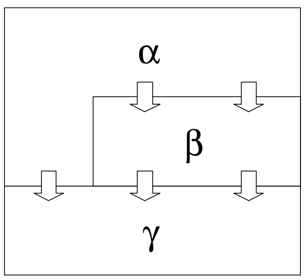

Figura 4.1: Pilha de protocolos do pequeno exemplo fictício

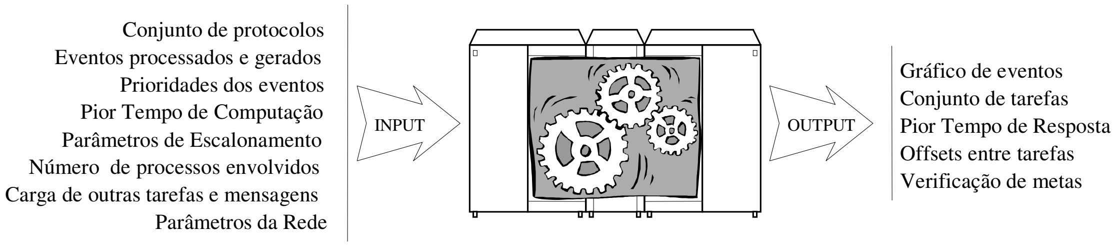

Figura 4.2: Diagrama do sistema proposto

ma composição de protocolos composta por três protocolos fictícios: Alfa, Beta e Gama (Fig. 4.1). Cada um deles apenas processa um evento, excepto o Beta que processa dois eventos. O Alfa gera três eventos, o Beta apenas um e o Gama não gera nenhum. De forma a simplificar o exemplo, não se consideram comunicações bidireccionais, nem se expõe nenhuma rede de comunicações (embora se possa considerar que por exemplo, o Beta encapsule uma).

4.1 Modelo do sistema

De modo a aplicar a análise descrita no capítulo anterior, é necessário introduzir algumas restrições aos sistemas que se pretendem estudar.

As tarefas estão atribuídas a um processador, sem possibilidade de migração. As prioridades de cada tarefa são fixas e atribuídas por uma entidade externa ao sistema (com uma excepção temporária aos casos de tarefas a beneficiarem de protocolos de herança de prioridade [32]). Entre cada activação de uma tarefa decorre um certo intervalo de tempo (o período), não podendo haver novas activações antes do fim deste intervalo. Os vários relógios dos computadores que fazem parte do sistema distribuído em análise não sofrem desvios relevantes entre si (o valor de precisão desta sincronização é um dos principais contribuintes para o *jitter*). Vários métodos de sincronização de relógios em sistemas distribuídos foram já propostos, sendo provável que a sua utilização tenha um certo impacto na análise dos sistemas que os utilizem, pois contribuem com uma certa carga de fundo nos processadores e na rede de comunicações.

Uma pilha de protocolos é constituída por várias camadas de protocolos. Cada protocolo a ser estudado é constituído por uma ou mais tarefas, podendo cada uma receber e processar apenas um evento, e em consequência deste, gerar zero ou mais eventos. Em consequência da recepção de um evento, existe um conjunto de eventos que podem ser gerados. À ordenação de forma causal de todos os eventos possíveis de ser gerados a partir de um dado evento, dá-se o nome de *cadeia de eventos*. Estes eventos podem ser recebidos no mesmo computador onde foram emitidos, tendo por isso um tempo de transmissão considerado nulo, ou serem transmitidos para um computador diferente através de uma rede de comunicações, havendo por isso um tempo de transmissão associado.

4.2 Pré-processamento

4.2.1 Dados necessários

A metodologia proposta opera sobre uma meta-descrição dos protocolos a serem processados. A forma exacta desta meta-descrição não é relevante, desde que a seguinte informação possa ser extraída:

O conjunto de protocolos utilizados De modo a poder fazer uma análise completa da pilha de micro-protocolos, é necessário determinar-se o conjunto de protocolos que a formam. Na versão actual da metodologia, este é um dos parâmetros fornecidos, embora seja possível conceber o algoritmo necessário a uma automatização deste processo. Esta seria feita de forma recursiva a partir da lista de dependências dos protocolos das camadas mais elevadas, até atingir a camada mais baixa da pilha.

Exemplo No exemplo fictício, o conjunto de protocolos utilizados é: Alfa, Beta e Gama.

O conjunto de eventos processados por cada camada, e os eventos gerados em consequência Cada micro-protocolo pode ser capaz de processar um ou mais eventos, alguns devido a pedidos da camada superior (ou da camada aplicacional), outros com origem nas camadas inferiores. Cada um destes eventos recebidos pode ou não originar múltiplos eventos em consequência. De reparar que estes conjuntos podem facilmente constituir a lista de dependências que permita a execução de um algoritmo para descobrir qual o conjunto de protocolos utilizados e as relações de precedência entre eles.

O conjunto de eventos processados e os vários conjuntos de eventos gerados são actualmente fornecidos, embora a sua determinação pudesse ser feita de forma automática. No entanto, esta não é trivial, sendo necessária uma análise de uma meta-descrição dos protocolos. Esta análise pode ser mais ou menos inteligente, de forma a distinguir eventos opcionais (quando, no algoritmo do micro-protocolo, a execução é condicionada, podendo ou não ser dada origem a um subconjunto de todos os eventos possíveis de serem gerados). De acordo com a complexidade desejada, esta ferramenta poderá ser desde um simples analisador sintáctico de detecção do envio ou recepção de eventos, até verdadeiro compilador de protocolos que além de gerar o código objecto para a arquitectura final gera também estes conjuntos (esta última aproximação costuma ser habitual em sistemas de determinação do pior tempo de execução).

Exemplo Os eventos processados e gerados são então os seguintes:

- Alfa: ALFA.REQ (gera BETA.UM, BETA.DOIS, GAMA.REQ);
- Beta: BETA.UM (gera GAMA.REQ) e BETA.DOIS (gera GAMA.REQ);
- Gama: GAMA.REQ.

A prioridade de cada evento Como é habitual em sistemas de tempo real, pretende-se que o processamento de certos eventos seja prioritário em relação a outros, através da atribuição de prioridades aos eventos. Como em termos de execução os eventos são apenas mensagens transferidas entre diferentes tarefas, a sua diferente prioridade de tratamento está dependente da prioridade das tarefas que os processam. Assim sendo, a prioridade de cada evento, atribuída na especificação, será transmitida à tarefa que o processa de forma directa (i.e. quanto mais prioritário for o evento maior será a prioridade da tarefa que o trata). Ao longo de uma cadeia de eventos, os eventos gerados herdam a prioridade daquele que lhe deu origem (devido a ser esta a única forma de garantir que o evento mantém a prioridade atribuída).

Exemplo A prioridade de todos os eventos será a que for dada ao evento ALFA.REQ, visto que este está na origem dos restantes.

4.2.2 Parâmetros específicos

Além desta informação independente do sistema a ser concretizado, existem também uma série de parâmetros que dependem do ambiente em que a pilha de protocolos irá ser executada:

O período de cada evento recebido Um dos mais importantes, este parâmetro especifica de quanto em quanto tempo é esperado um novo evento de um dado tipo. De notar que mesmo os eventos esporádicos, ou sem um período definido, são possíveis de analisar desde que aqui seja dada a máxima taxa de repetição do evento (no caso do cenário resultante ser demasiado pessimista, é sempre exequível reduzir esta taxa). Associado a este parâmetro, está o valor do *jitter*, que quantifica a máxima variação entre o instante em que uma dada acção devia ocorrer e o instante em que ela efectivamente ocorre.

O pior tempo de computação das tarefas Um parâmetro de difícil determinação, o pior tempo de computação das tarefas, está intimamente ligado à arquitectura do sistema a ser analisado. O ideal seria poder obter este valor a partir de uma análise do algoritmo do protocolo, no entanto esta tarefa é muitíssimo complexa e foge ao âmbito deste trabalho (várias soluções para este problema foram propostas, desde linguagens de programação específicas em que todos os ciclos são limitados, compiladores adaptados que geram a informação necessária, até modelos extremamente sofisticados dos processadores [27]). De forma a obter o valor para o pior tempo, são geralmente realizadas um grande número de execuções das tarefas na arquitectura final do sistema, e é usado o pior tempo obtido durante esses testes. Obviamente, isto requer que os testes tenham coberto todas as diferentes rotas de execução dentro da tarefa, o que nem sempre é fácil de garantir em tarefas mais complexas.

Os parâmetros de escalonamento Estes parâmetros, frequentemente ignorados pelas análises teóricas, embora sejam muito relevantes em termos práticos, são os períodos de activação do próprio escalonador e o tempo dispendido por este a realizar as operações necessárias a colocar as diversas tarefas suspensas, mas prontas a correr, na fila de execução. Este último valor é também obtido por medição de vários testes, embora para alguns sistemas mais vulgares seja informação publicamente disponível.

O número de processos que participam na comunicação Em protocolos de comunicação que não sejam ponto-a-ponto, ou seja, em que participam mais de 2 nós, é preciso saber qual o número desses nós, pois isso pode implicar um tratamento diferente dos eventos a serem estudados (por exemplo, recepção de mais mensagens de confirmação).

De igual modo, para protocolos em que há resposta por parte do receptor, é preciso saber qual o período das tarefas emissoras dessas respostas.

A carga de outras tarefas e mensagens Além das tarefas envolvidas na pilha de protocolos a ser estudada, podem existir outras tarefas que também usam os recursos do sistema (tempo de processamento e acesso a zonas de exclusão mútua). Para que a análise seja rigorosa, é necessário incluir também estas últimas. O mesmo se passa com a emissão de mensagens na rede de comunicações.

Os parâmetros da rede de comunicações Devido à sua especificidade, a rede de comunicações não é tratada do mesmo modo que as outras camadas da pilha de protocolos. Deste modo, parte do trabalho da análise de escalonabilidade consiste em desenvolver e integrar um modelo formal da rede de comunicações. Este modelo terá também alguns parâmetros associados, como largura de banda, latência do canal, probabilidade de erros, tamanho máximo da mensagem, etc. A possibilidade de haver múltiplas redes também deve ser tomada em conta, podendo ser usados modelos diferentes no caso em que estas não sejam idênticas.

4.3 Informação gerada

4.3.1 Grafo de eventos

O grafo de eventos é a estrutura construída de forma a facilitar o processamento dos passos seguintes da análise. Existirá um grafo de eventos para cada evento possível de ser gerado pela camada superior.

A construção do grafo de eventos usa a informação anteriormente recolhida dos conjuntos de eventos processados e dos conjuntos de eventos gerados em consequência.

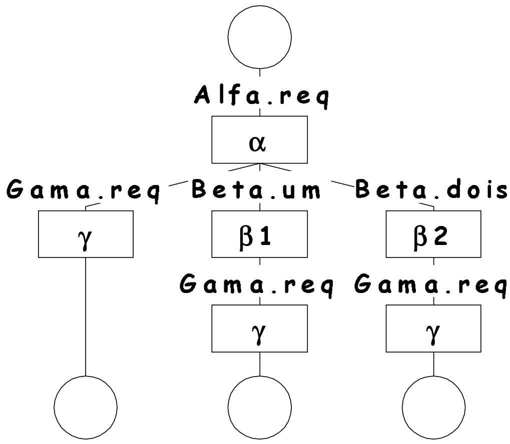

Figura 4.3: Grafo de eventos do pequeno exemplo fictício

Para cada um dos eventos tratáveis em resposta a pedidos vindos de camadas superiores, é construído um grafo de eventos diferente. Para cada um destes grafos, o evento recebido da camada superior está na raiz, tendo como nó a camada de protocolo que o trata. Este nó tem como folhas todos os eventos gerados em consequência do evento raiz desse nó. Este procedimento é aplicado de forma recursiva em cada camada, até que já não existam mais eventos gerados, ou que estes sejam entregues à camada superior.

Exemplo O grafo gerado está ilustrado na figura 4.3.

4.3.2 Conjunto de tarefas de protocolos

Partindo do grafo de eventos, é possível determinar quais as tarefas em que o protocolo deve ser dividido. Cada um dos nós do grafo de evento corresponde a uma tarefa, sendo que algumas podem ser repetidas ao longo do grafo. Como cada nó é apenas originado por um evento, a tarefa que corresponde a esse nó trata esse evento.

Exemplo Os protocolos do exemplo fictício deverão ser concretizados em apenas uma tarefa no caso dos protocolos Alfa e Gama, enquanto que o Beta deverá ter uma tarefa que processe o evento BETA.UM e outra para o BETA.DOIS.

4.3.3 Determinação dos piores tempos de resposta

Uma vez construído o grafo de eventos, pode-se construir um modelo do sistema a estudar, usando as relações de precedência entre as tarefas e os parâmetros fornecidos como dados de entrada para uma técnica de análise de escalonabilidade. Devido aos sistemas em estudo serem distribuídos em vários computadores, é necessário iterar repetidas vezes o ciclo até os resultados convergirem no valor final. O primeiro passo é a obtenção dos piores tempos de resposta das tarefas nos vários processadores, com os desfasamentos iniciados de forma a respeitar as relações de precedência e considerando o tempo de transmissão de todos os eventos como sendo o mínimo (i.e. sem interferências). De seguida, actualizam-se os valores dos desfasamentos das mensagens de forma a que estes correspondam ao pior tempo de resposta das tarefas que os geram, e aplica-se a análise a estas de acordo com o modelo da rede usada, obtendo-se o pior tempo de resposta destas mensagens. Estes são usados para actualizar o valor dos desfasamentos das tarefas que os recebem, sendo então repetido o passo de calcular o pior tempo de resposta de todas as tarefas. Como estes valores crescem de forma estritamente monótona, os valores acabam por convergir, ou então o sistema não é escalonável.

4.3.4 Determinação dos desfasamentos de tempo entre tarefas

Ao longo da aplicação da análise à pilha de protocolos, e obedecendo às relações de precedência entre as diferentes tarefas que tratam os eventos, é possível determinar também o desvio temporal, ou desfasamento, que vai afectar cada tarefa devido ao tempo de computação das tarefas que a precedem. Este desfasamento será a soma dos melhores tempos de computação das tarefas que precedem cada uma das tarefas dentro de uma cadeia de eventos. Desta forma, pode-se evitar o desperdício de recursos do sistema, ao colocar apenas as tarefas na fila de execução quando já existirem condições para a recepção do evento que elas tratam. Por outro lado, pode-se melhorar o tempo de resposta do sistema, ao colocar na fila de execução as tarefas exactamente na altura em que o evento que elas tratam é recebido.

4.3.5 Verificação das Metas

Após a aplicação da análise, obter-se-á para cada uma das tarefas o seu pior tempo de resposta. Este pior tempo de resposta será comparado com a meta desejada de forma a verificar se esta é cumprida. Num sistema em que se pretende que as metas nunca seja ultrapassadas, pode-se proceder a alterações de forma a resolver os conflitos encontrados. Algumas possibilidades serão: a alteração das prioridades dos eventos, o alívio da carga de fundo e/ou a redução dos tempos de computação das tarefas (através de mais velocidade de processamento ou de uma melhoria do código das tarefas).

4.4 Sumário

Foi apresentada neste capítulo uma metodologia que permite aplicar a técnica de escalonabilidade anteriormente descrita no cap. 3. Esta metodologia permite tirar partido das particularidades dos sistemas de composição de micro-protocolos, visando deste modo reduzir algum do pessimismo. Foram descritos os pré-requisitos que precisam de ser fornecidos, e os resultados obtidos ao aplicar a metodologia proposta. Foi usado um pequeno exemplo de modo a melhor ilustrar esta aplicação. No capítulo seguinte, será aplicada esta técnica a um sistema de protocolos modulares existente, o RELCAN e o EDCAN sobre o CAN, e obtidos alguns resultados do pior tempo de resposta destes.

Capítulo 5

Aplicação Prática: o RELCAN

O protocolo RELCAN foi desenvolvido por Rufino et al. em [31], como forma de resolver uma falha do barramento de campo CAN. Este protocolo fornece garantias de difusão fiável, sendo apoiado no protocolo EDCAN de difusão ávida. Estes protocolos foram desenvolvidos de raiz como protocolos modulares, com o fim de serem usados em composições de protocolos em ambientes de tempo-real estrito. Devido à sua adequação aos objectivos do sistema de análise de protocolos desenvolvido neste trabalho, foi escolhido como caso prático para aplicação da técnica proposta. Este capítulo começa por fornecer uma breve introdução teórica do barramento CAN, seguido da apresentação do seu modelo teórico. Antes da aplicação da técnica ao RELCAN é feita uma breve apresentação do funcionamento do EDCAN e RELCAN.

5.1 Breve introdução ao CAN

O CAN (Controller Area Network) [13, 30] é um barramento de comunicação desenvolvido inicialmente para a industria automóvel, mas que tem encontrado aplicações noutras áreas da computação em tempo real.

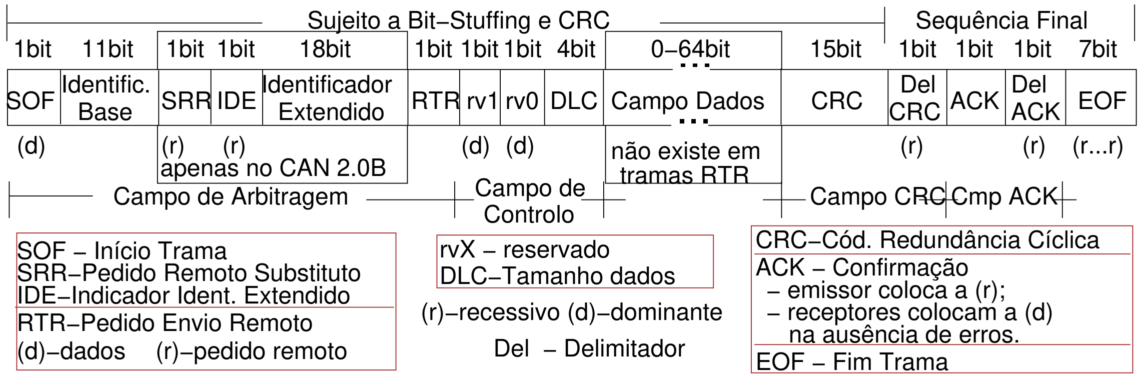

Figura 5.1: Trama do CAN

Existem dois tipos de tramas no CAN, o tipo definido originalmente com identificadores de 11 bits (CAN 2.0A) e a extensão que usa identificadores de 29 bits (CAN 2.0B). Ambos os tipos estão esquematizados na Fig. 5.1, onde é possível ver cada um dos campos da trama completa. A camada física do CAN força a sincronização de todos os nós da rede e faz com que haja apenas um único bit em qualquer altura no barramento.

A codificação é feita através de bits dominantes (0) e recessivos (1). Um bit recessivo só aparece no barramento quando todos os nós transmitem bits recessivos, bastando haver apenas um nó a transmitir um bit dominante para que este prevaleça. Cada nó espera que o CAN esteja inactivo para começar a transmitir uma nova trama, podendo haver vários nós a iniciar a transmissão simultaneamente. Cada mensagem tem o seu identificador, sendo que quanto menor este for, maior é a prioridade da mensagem. Em caso de transmissão simultânea cada nó irá tentar transmitir a sua mensagem, até que durante a transmissão do identificador, a mensagem com maior prioridade (com o menor identificador) se impõe sobre as outras devido à transmissão de um bit dominante, enquanto as de menor prioridade emitem um bit recessivo. Após a transmissão do identificador, apenas um nó estará a transmitir, excepto se a mensagem for um pedido de transmissão remota (caso em que vários nós podem transmitir simultaneamente).

Existem dois tipos de mensagem no CAN: as mensagens de dados e as mensagens de pedido de transmissão remota (RTR). Estas últimas são usadas para solicitar a transmissão de determinada mensagem e contêm um campo de dados nulo. A transmissão simultânea de várias mensagens RTR é possível uma vez que estas são idênticas bit a bit, não sendo possível o emissor verificar se é o único no barramento. As mensagens de dados podem conter até um máximo de 64 bits de dados.

O CAN define ainda que não pode haver mais de 5 bits idênticos consecutivos dentro do conteúdo da trama (excepto os últimos 10 bits), de modo a manter a sincronia entre os nós. Quando, devido ao conteúdo dos dados, esta regra for violada são inseridos bits extra de modo a evitar este problema (*bit-stuffing*). Quando é detectada a transmissão de 6 bits idênticos seguidos, é accionado o mecanismo de sinalização de erros. Existem alguns delimitadores e sinalizadores de erro que violam a regra dos 6 bits, mas estas situações são criadas de propósito para que um erro não detectado por um nó seja obrigatoriamente detectado durante a transmissão do sinal de erro.

5.2 Análise do tempo de resposta do CAN

Para utilizar a análise de escalonabilidade no CAN é preciso dispor-se de um modelo que permita obter o pior tempo entre o envio e a recepção das mensagens. Este modelo é facilitado pelas semelhanças existentes entre o CAN e um processador, no que diz respeito ao escalonamento. Tal como num escalonador de computador, existem vários níveis de prioridade e as mensagens podem começar a ser transmitidas assim que são enviadas, desde que não haja nenhuma mensagem a ocupar o barramento nesse instante e não haja nenhuma mensagem com maior prioridade à espera de ser transmitida. O modelo apresentado nesta secção foi desenvolvido por Tindell em [35, 36].

5.2.1 Análise de um modelo simples do CAN

Tal como em análises anteriores, as mensagens são consideradas periódicas, com a mensagem m tendo um período T_m. Existe um *jitter* associado à colocação das mensagens na fila de envio, sendo este J_m. A mensagem m ocupa b_m bytes e tem C_m como pior tempo de transmissão no barramento (sendo C_m proporcional a b_m).

O pior tempo de resposta de uma mensagem m é o maior tempo entre a colocação da mensagem na fila de envio e a sua recepção nos nós de destino e é dado por R_m. A meta para que a mensagem chegue ao destino é dada por D_m. Pretende-se que $R_m \leq D_m$. No CAN, se uma mensagem não for enviada antes da seguinte com o mesmo identificador, é apagada por esta, o que leva à restrição adicional: $R_m \leq T_m - J_m$.

O tempo de resposta da mensagem m é constituído por dois factores: o tempo que fica na fila à espera de ser colocada no barramento (t_m) e o tempo que demora a ser transmitida (C_m).

$$R_m = t_m + C_m \tag{5.1}$$

O tempo que a mensagem fica na fila de espera depende do tempo de bloqueio por mensagens de menor prioridade que estejam a ocupar o barramento nessa altura (B) e do tempo de transmissão de outras mensagens de maior prioridade. Este tempo é dado pela seguinte equação:

$$t_m = B + \sum_{\forall j \in hp(m)} \left\lceil \frac{t_m + J_j + \tau_{bit}}{T_j} \right\rceil C_j \tag{5.2}$$

onde τ_{bit} é o tempo que demora a transmitir um bit no CAN (para uma largura banda de 1Mbps, $\tau_{bit} = 1\mu s$). É possível estabelecer um valor máximo para B, assumindo a maior mensagem possível do CAN, e com o máximo de *bit-stuffing* o que dá um valor de $157\tau_{bit}$ [30].

5.2.2 Extensão para recuperação de erros

O CAN prevê, além da normal transmissão de tramas, a emissão de sinais de erro que podem levar à interrupção momentânea de transmissões no barramento. Para lidar com estas situações é preciso incluir na equação anterior um factor E que maximize o tempo dispendido na recuperação do erro.

$$t_m = E(t_m + C_m) + B + \sum_{\forall j \in hp(m)} \left\lceil \frac{t_m + J_j + \tau_{bit}}{T_j} \right\rceil C_j \tag{5.3}$$

Este factor E pode ser determinado através de medidas feitas ao CAN na situação a estudar, ou através de um modelo estatístico. Um modelo simples é dado por:

$$E_m(t) = \left(n_{error} + \left\lceil \frac{t}{T_{error}} \right\rceil - 1 \right) \left(29\tau_{bit} + \max_{\forall k \in hp(m) \cup \{m\}} C_k \right) \tag{5.4}$$

Nesta equação, o primeiro termo indica o número máximo de erros que podem ocorrer no intervalo t, sendo n_{error} o número de erros que podem ocorrer num intervalo arbitrariamente pequeno e T_{error} o intervalo de tempo após os n_{error} iniciais durante o qual não podem aparecer mais erros, e depois apenas 1 por cada T_{error}. O segundo termo da equação indica o tempo máximo que cada erro pode demorar a ser recuperado. Cada erro pode originar, no máximo, 29 bits em sinalizações do erro, demorando então $29\tau_{bit}$. O erro só se considera recuperado após a correcta transmissão da mensagem m original, pelo que é preciso entrar em conta com o tempo de transmissão desta e de outras mensagens de maior prioridade que entretanto surjam no barramento.

5.2.3 Trabalho relacionado

O modelo de erros apresentado na equação 5.4 foi estendido por Punnekkat et al. em [26, 25]. No seu modelo geral de falhas são incluídas múltiplas fontes de erro e são considerados padrões de erros (de forma a simular telefones celulares, radares, etc.). Em [23] Pinho et al. estendem o modelo simples de Tindell (eq. 5.2) de forma a incluir os tempos de inacessibilidade do CAN devido a erros do próprio canal e dos *transceivers* dos dispositivos conectados.

5.3 Falhas na detecção de erros no CAN

O CAN, tal como está definido, aparenta ser capaz de assegurar difusões atómicas. No entanto, em [31] é demonstrado que pode haver erros que dão origem a possíveis inconsistências, levando a que alguns nós aceitem uma mensagem e outros não, podendo depois haver transmissão de duplicados, caso o emissor ainda continue a funcionar depois do erro ser detectado.

Uma das situações que dá origem a este problema verifica-se quando alguns dos receptores detectam um bit dominante durante a transmissão do campo EOF (End Of Frame), que é constituído por 7 bits recessivos. Se o bit dominante for o último bit deste campo, a norma do CAN diz que a trama deve ser aceite pois não existe forma de saber se os outros receptores também detectaram o erro. Nesta situação, todos os receptores aceitam a trama. Mas, se um bit dominante for detectado pelo menos por um receptor no penúltimo bit do EOF, então esse receptor iniciará a transmissão de um sinal de erro e rejeitará a trama. Os outros receptores que não tenham detectado o erro poderão detectar o sinal de erro, mas terão de aceitar a trama tal como foi descrito antes. Isto leva a que alguns nós recebam a mensagem e outros não. Caso o emissor da mensagem detecte o erro, irá proceder à retransmissão, havendo alguns nós que irão receber a mesma mensagem duas vezes.

Para colmatar esta falha, Rufino et al. desenvolveram então alguns protocolos para fornecer serviços de difusão fiável (RELCAN) e também serviços de difusão totalmente

```
Sender
ES1   when edcan.req(mid⟨type,p,n⟩, mess) invoked at p do
          if mess = NULL then
              can-rtr.req(mid);
          else
              can-data.req(mid, mess);
      od;

ES2   when can-rtr.cnf(mid)
      or can-data.cnf(mid, mess) confirmed do
          deliver edcan.cnf (mid,mess);
      od;

Recipient
Initialization
init  ndup(mid) := 0; // number of duplicates, kept for each message

ER    when can-data.ind(mid, mess) received at q
      or can-rtr.ind(mid, mess=NULL) received at q do
          ndup(mid) := ndup(mid) + 1;
          if ndup(mid)= 1 then // new message
              edcan.ind (mid, mess);
              if mess = NULL then
                  can-rtr.req(mid); // clustered
              else
                  can-data.req(mid, mess);
              fi;
          elif ndup(mid) > j then
              can-abort.req(mid);
          fi;
      od;
```

Figura 5.2: Protocolo de Difusão Ávida

ordenada (TOTCAN), ambos assentes sobre um protocolo de difusão ávida (EDCAN). Neste trabalho não será abordado o TOTCAN.

5.4 O protocolo EDCAN

O protocolo EDCAN, definido na Fig. 5.2, funciona por difusão ávida. Os receptores retransmitem a mensagem após a recepção, de forma a garantir que não existem mensagens recebidas apenas por alguns nós e que não sejam recebidas por todos.

Após a recepção do pedido de envio de mensagem através do EDCAN, é feito o envio dessa mensagem através da camada CAN. Caso a mensagem contenha dados, é

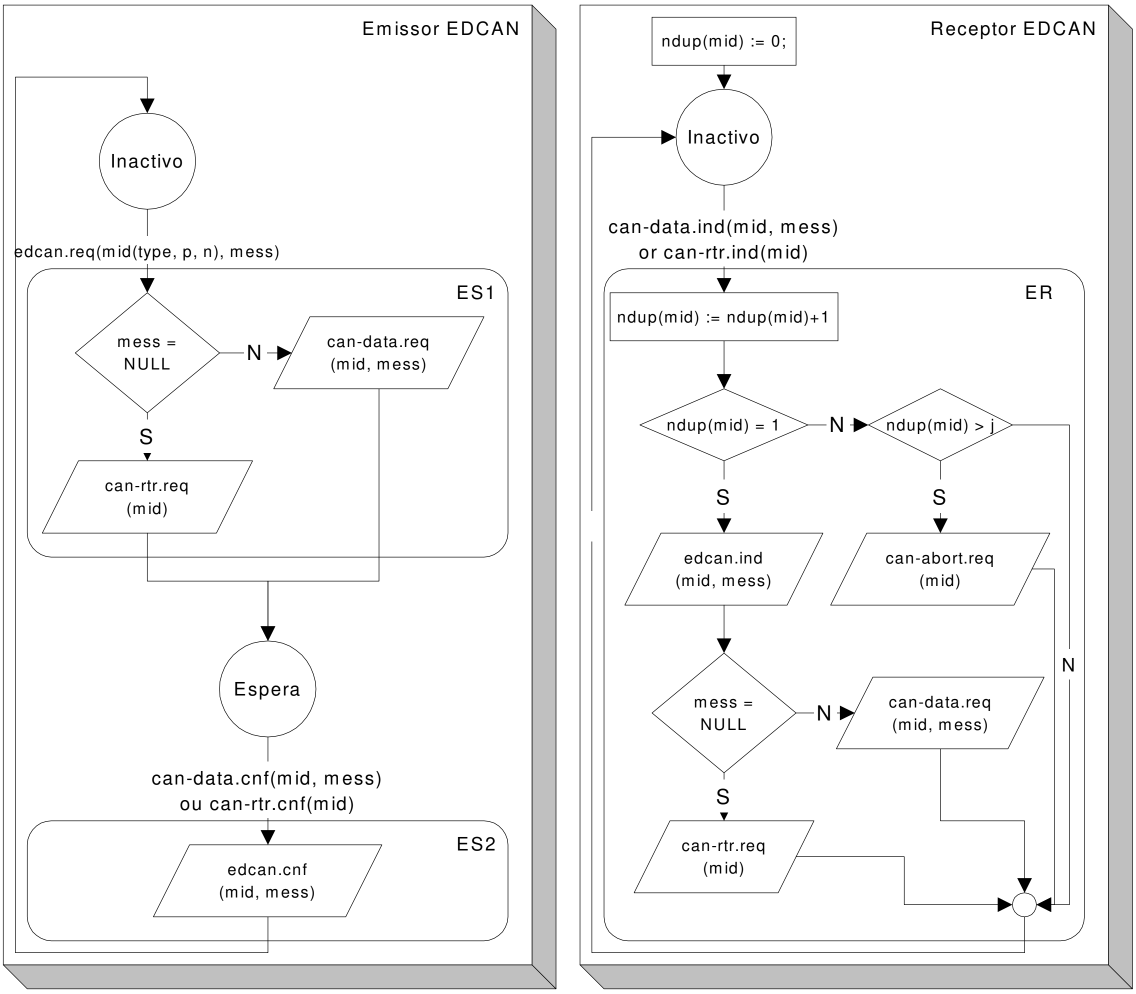

Figura 5.3: Diagrama do EDCAN

usada uma trama de dados, caso contrário é usada uma trama RTR, permitindo assim várias transmissões simultâneas. Se o emissor não falhar, a mensagem é entregue com sucesso.

O EDCAN baseia o seu funcionamento nas retransmissões que os receptores fazem após receberem uma mensagem pela primeira vez. Estas retransmissões também usam tramas de dados ou RTR, conforme a mensagem contenha dados ou não. Caso a mensagem não contenha dados, o uso de tramas RTR permite poupar algum tempo, visto que algumas retransmissões serão feitas em simultâneo. Após um certo número j de retransmissões detectadas, considera-se que a mensagem terá sido entregue a todos os receptores, pelo que os receptores que ainda não tenham enviado a sua retransmissão tentarão cancelá-la.

5.5 O protocolo RELCAN

Sendo baseado no EDCAN, o RELCAN aproveita os serviços de difusão ávida, mas utiliza-os apenas quando estritamente necessário.

O emissor começa por atribuir um identificador único à mensagem, baseado no identificador do nó e na variável local rel_sn. O emissor inicia então a primeira das duas fases do protocolo. Na primeira fase, a mensagem é enviada usando o serviços do CAN, e o emissor aguarda a respectiva confirmação por parte do controlador do CAN. Ao recebê-la, o emissor tem a certeza de que todos os receptores receberam a mensagem e, executando a segunda fase, emite uma mensagem de confirmação.

Os receptores começam por entregar a primeira cópia da mensagem, e iniciam um alarme. Ao receber a confirmação por parte do servidor, o receptor cancela esse alarme. Caso o alarme expire antes da confirmação chegar, procede-se à difusão ávida da mensagem usando o EDCAN. Ao receber a primeira das retransmissões por parte do

```
Sender
init   rel_sn := 0; // local sequence number

RS1    when relcan.req(mess) invoked at p do
           rel_sn := rel_sn + 1;
           can-data.req(mid⟨R-DATA,p,rel_sn⟩, mess);
       od;

RS2    when can-data.cnf(mid⟨R-DATA,p,rel_sn⟩, mess) received do
           can-rtr.req (mid⟨CONFIRM,p,rel_sn⟩);
           relcan.cnf (mess);
       od;

Recipient
init   ndup(mid) := 0; // number of duplicates, kept for each message
       data(mid) := NULL; // data part of the message

RR1    when can-data.ind(mid⟨R-DATA,p,n⟩, mess) received at q do
           ndup(mid) := ndup(mid) + 1;
           data(mid) := mess;
           start alarm (mid);
           if ndup(mid)= 1 then // new message
               relcan.ind (mess);
           fi;
       od;

RR2    when can-rtr.ind(mid⟨CONFIRM,s,n⟩) received at q do
           data(mid) := NULL;
           cancel alarm(mid);
       od;

RR3    when alarm(mid) expires at q do
               edcan.req (mid, data(mid));
       od;

RR4    when edcan.ind(mid⟨R-DATA,p,n⟩, mess) received at q do
           ndup(mid) := ndup(mid) + 1;
           if ndup(mid)= 1 then // new message
               relcan.ind (mess);
           fi;
       od;
```

Figura 5.4: Protocolo de Difusão Fiável

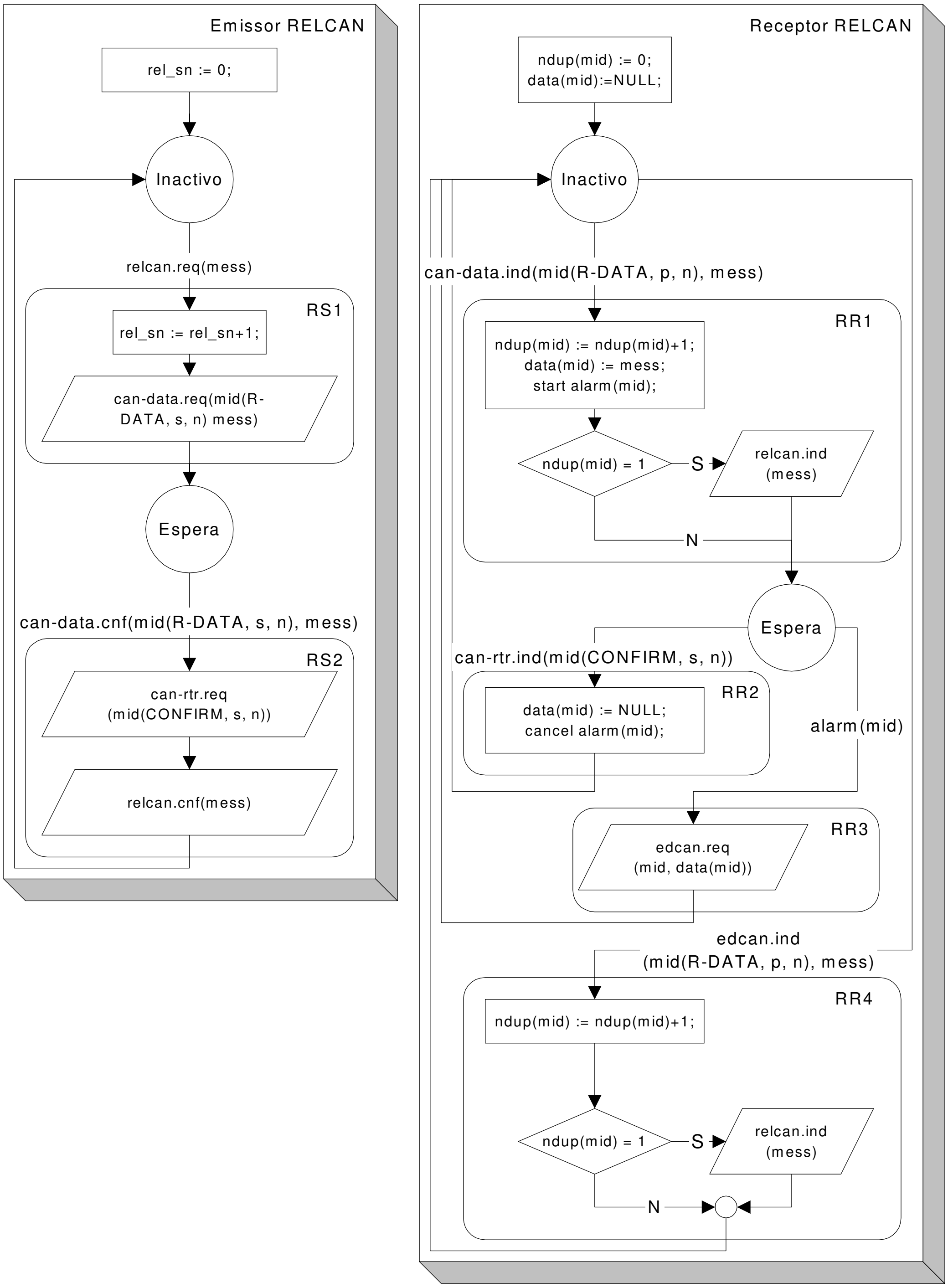

Figura 5.5: Diagrama do RELCAN

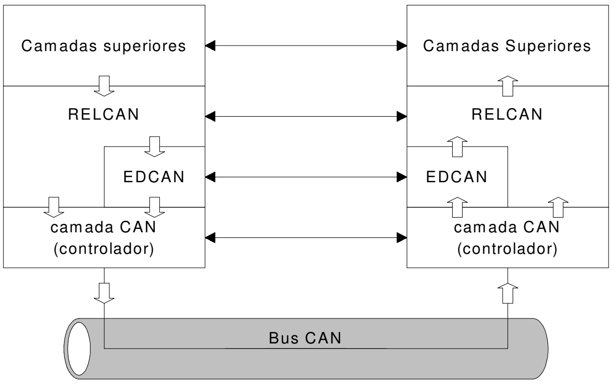

Figura 5.6: Pilha de protocolos do RELCAN

EDCAN, o receptor entrega então a mensagem.

Este protocolo tem a vantagem de não despoletar a retransmissão de mensagens excepto quando estritamente necessário, caso em que o seu desempenho se aproxima do EDCAN.

5.6 Análise do RELCAN

Tendo apresentado a rede CAN e os protocolos EDCAN e RELCAN, é chegada a altura de aplicar a metodologia proposta a alguns cenários desenvolvidos com base nestes protocolos, começando por gerar os seus grafos de eventos. O primeiro cenário desenvolvido permitirá obter manualmente alguns valores de escalonabilidade de RELCAN, que permitem uma melhor compreensão do funcionamento deste. Os cenários seguintes serão cenários mais complexos, que envolvem interferências de tarefas de diferentes níveis de prioridade.

5.6.1 Conjunto de protocolos

A pilha de protocolos do RELCAN é constituída pelos protocolos RELCAN, EDCAN e CAN (Fig. 5.6).

5.6.2 Conjunto de eventos processados e gerados

5.6.2.1 RELCAN

O emissor RELCAN (Fig. 5.4) gera dois eventos para a camada CAN (CAN-DATA.REQ e CAN-RTR.REQ) e um para a camada superior (RELCAN.CNF), e processa um evento da camada superior (RELCAN.REQ) e um da camada CAN (CAN-DATA.CNF).

O receptor (Fig. 5.4) gera um evento para a camada superior (RELCAN.IND), um evento para a camada EDCAN (EDCAN.REQ) e dois eventos para o serviço de alarme (ALARM.START e ALARM.CANCEL). Pode processar dois eventos da camada CAN (CAN-DATA.IND e CAN-RTR.IND), um evento do alarme (ALARM.TIMEOUT) e um evento do EDCAN (EDCAN.IND).

5.6.2.2 EDCAN

O emissor EDCAN (Fig. 5.2) gera um evento para a camada superior (EDCAN.CNF) e dois eventos para a camada CAN (CAN-DATA.REQ e CAN-RTR.REQ), processa um evento da camada superior (EDCAN.REQ) e dois eventos da camada CAN (CAN-DATA.CNF e CAN-RTR.CNF).

O receptor (Fig. 5.2) gera um evento para a camada superior (EDCAN.IND) e três eventos para a camada CAN (CAN-DATA.REQ, CAN-RTR.REQ e CAN-ABORT.REQ). Processa dois eventos da camada CAN (CAN-DATA.IND e CAN-RTR.IND).

5.6.3 Conjunto de tarefas de protocolos

5.6.4 Pior Tempo de Computação

Idealmente, o pior tempo de computação seria obtido por medições das tarefas de uma concretização do RELCAN e EDCAN, mas como ainda não existe nenhuma disponível,

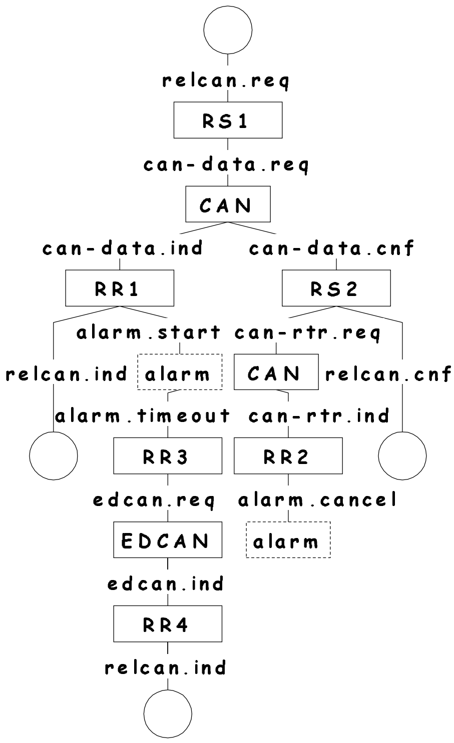

Figura 5.7: Grafo de Eventos do RELCAN

teve de ser usado um valor fictício. Para os casos assumidos a seguir, usou-se como pior tempo de computação o valor de $150\mu s$. Este está de acordo com os habitualmente encontrados na literatura para tarefas simples. Uma concretização do RELCAN pode, no entanto, ficar sujeita a perdas de mensagens, no caso do pior tempo de computação ser maior do que o tempo mínimo de transmissão de mensagens no CAN ($44\mu s$), quando a transmissão de mensagens for muito intensiva.

5.6.5 Grafo de eventos

5.6.5.1 RELCAN

O grafo de eventos do RELCAN (ver Fig. 5.7), começa com o único evento recebido da camada superior (RELCAN.REQ), sendo possível observar a cadeia de eventos do melhor caso (a mensagem de CONFIRM é enviada, e o alarme cancelado, no lado direito) e do

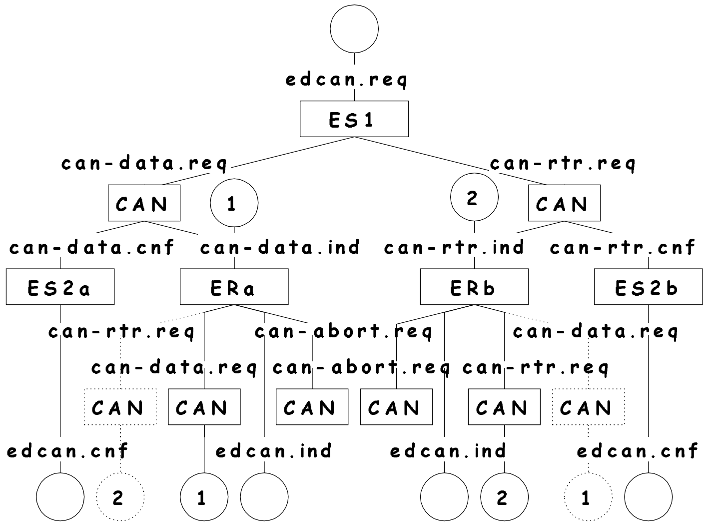

Figura 5.8: Grafo de Eventos do EDCAN

pior caso (em que o alarme é activado, no lado esquerdo). A análise destes dois casos será objecto das secções seguintes.

5.6.5.2 EDCAN

O grafo de eventos do EDCAN (ver Fig. 5.8) apresenta uma simetria em relação à mensagem ser ou não vazia. O carácter recursivo do protocolo é evidenciado no facto de o receptor EDCAN tratar um evento que é gerado em consequência de um evento gerado por si. As linhas a tracejado na figura representam eventos gerados pelo receptor EDCAN, incluídos numa análise simples, que seriam excluídos com o uso de uma análise mais complexa com conhecimento de contexto.

5.6.6 Caso Simples

Para fazer a análise temporal do RELCAN, é conveniente começar por um caso simples em que não se incluem efeitos de escalonamento, bloqueamento, *jitter* e interferência. Este caso simples permite-nos fazer os cálculos manualmente de forma a obter uma

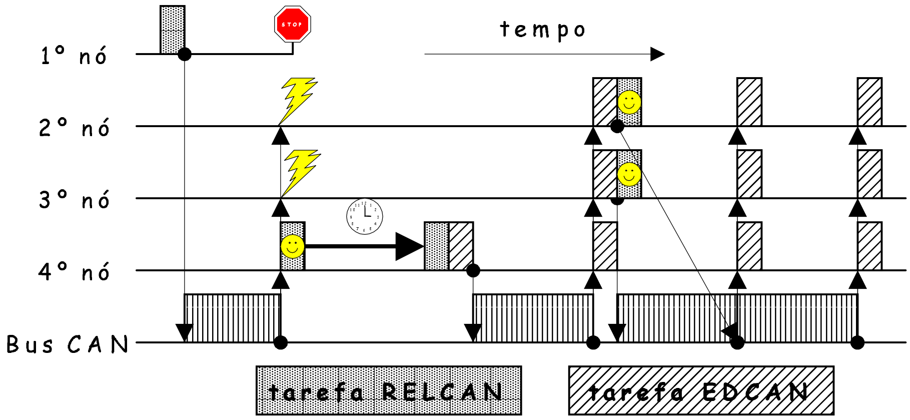

Figura 5.9: Cenário simples do RELCAN: barramento com 4 nós

melhor percepção do funcionamento do protocolo. O cenário desenvolvido consiste em quatro nós no barramento CAN sendo que apenas um nó transmite, este nó falha após a transmissão da mensagem e antes de enviar a mensagem de confirmação (ver Fig. 5.9).

5.6.6.1 Pior Tempo de Resposta

O pior tempo de resposta para uma mensagem ser transmitida usando o protocolo RELCAN acontece quando o emissor falha, após enviar a mensagem com os dados mas antes de poder enviar a mensagem de CONFIRM, forçando a activação da camada EDCAN. Este valor é obtido a partir dos seguintes cálculos:

$$\begin{aligned} &C_{RS1} + C_m + C_{RR1} + alarm + C_{RR3} + \\ &C_{ES1} + C_m + C_{ER} + \max(C_{RR4}, C_m) + (j-2)\max(C_m, C_{ER}) + C_{ER} \end{aligned} \quad (5.5)$$

As parcelas desta soma contabilizam as várias contribuições para o pior tempo de resposta: o tempo de computação da tarefa que envia a mensagem de dados original, a transmissão dessa mensagem, a computação da tarefa que a recebe e inicia o alarme, o tempo definido até o alarme expirar, a computação das tarefas RELCAN e EDCAN que começam a retransmissão, a retransmissão dessa mensagem, a computação da tarefa que a trata, seguido pelo máximo entre o tempo de computação da tarefa RELCAN que entrega a mensagem à camada superior e o tempo de transmissão da mensagem, e finalmente o tempo que demora a tratar todas as seguintes retransmissões (máximo entre o tempo de transmissão e o tempo de computação da tarefa receptora), sendo a última parcela o tempo de computação da tarefa que trata a última retransmissão.

Usando os valores apresentados em [30], uma mensagem de CAN 2.0B pode demorar um máximo de $157\mu s$ a ser transmitida, num barramento de 1Mbps. Usando este valor, e o valor de $150\mu s$ para o pior tempo de computação das tarefas mencionado anteriormente, sendo o valor de expiração do alarme $400\mu s$, obtemos o valor de $1928\mu s$ para a equação 5.5, com j=3 nós no barramento (o emissor, por ter falhado, já não é considerado).

Para obter um termo de comparação, façamos os cálculos para o melhor caso:

$$C_{RS1} + C_{data} + C_{RS2} + \max(C_{rtr}, C_{RR1}) + C_{RR2} \tag{5.6}$$

com a mensagem CAN mais longa, o valor obtido é $757\mu s$, claramente melhor do que o valor obtido para o pior caso. No melhor caso, o protocolo EDCAN não chega a ser activado, pelo que o valor obtido é apenas a soma das tarefas que enviam e processam as mensagens de dados e de confirmação. Assume-se que a tarefa RS2 é de maior prioridade que RR1. A transmissão da mensagem de confirmação e a computação da tarefa de recepção da mensagem de dados são simultâneas, pelo que apenas se usa o maior destes valores (o tempo de transmissão de uma mensagem RTR varia entre o mínimo de $44mus$ até ao máximo de $77\mu s$ [30]).

5.6.6.2 Determinação dos desfasamentos de tempo entre tarefas

As equações 5.5 e 5.6, podem ser usadas para a determinação dos desfasamentos óptimos para activação das várias tarefas que compõem os protocolos.

5.6.6.3 RELCAN

Assumindo um desfasamento de $0\mu s$ para a tarefa RS1, o desfasamento para RS2 e RR1 que começam a execução ao mesmo tempo (no instante da recepção da mensagem de dados) será $194\mu s$. RR2 deverá ter um desfasamento de $388\mu s$, RR3 de $744\mu s$ e RR4 de $1238\mu s$.

5.6.6.4 EDCAN

Os desfasamentos para as tarefas do EDCAN são calculados assumindo um desfasamento de $0\mu s$ para a tarefa ES1 (sendo activado a partir do RELCAN, será $894\mu s$). Assim, a recepção da transmissão original ocorre $194\mu s$ depois, sendo este o valor do desfasamento de ES2. Todas as retransmissões são depois processadas com desfasamentos de $(388 + n.194)\mu s$.

5.6.6.5 Verificação das metas

Num sistema que use o barramento CAN, devido a cada nova mensagem apagar qualquer mensagem anterior com o mesmo identificador, é útil que o processamento das mensagens esteja terminado antes da nova mensagem ser emitida. Para que isso aconteça, basta que a meta da cadeia de eventos que é originada por essa mensagem seja igual ou inferior à periodicidade da mesma. Com base nos cálculos anteriores, é possível dizer que a meta mínima possível para uma mensagem RELCAN é $644\mu s$ (menor mensagem sem nenhum erro). Caso se permita a falha do emissor então é preciso que a meta seja maior que $1538\mu s$ (menor mensagem e máxima sobreposição das retransmissões).

5.6.7 Caso Complexo

O cenário desenvolvido anteriormente, embora permita fazer os cálculos manualmente, não permite obter uma visão do que acontece se várias cadeias de eventos RELCAN estiverem simultaneamente activas no mesmo barramento. De forma a estudar a interferência provocada por múltiplas cadeias de eventos foram desenvolvidos cenários mais complexos, cujos resultados foram obtidos através do uso de uma ferramenta especialmente adaptada para este fim.

5.6.7.1 Cenário Base

Este cenário consiste em 6 nós no mesmo barramento CAN, com 3 dos nós como emissores, sendo todos receptores. Todas as tarefas e mensagens da primeira cadeia de eventos têm maior prioridade que as das outras duas, sendo que destas uma é de maior prioridade que a outra. Os seguintes valores foram aplicados: $T_{clk} = 1\mu s$, $C_{clk} = 0\mu s$, $C_{QL} = 0\mu s$, $C_{QS} = 0\mu s$ (nenhum efeito do escalonador), período $T_{ti} = 3000\mu s$ para todas as transacções e $C_i = 150\mu s$, $e_i = 1$ (transmissão da mensagem em todos os períodos), $D_i = T_{ti=trans(i)} = 3000\mu s$ (meta igual ao período), e nenhum *jitter* ($J_i = 0\mu s$) para todas as tarefas. As mensagens de dados demoram $157\mu s$ a transmitir e as de confirmação, $64\mu s$. Note-se que o mesmo desfasamento em tarefas de transacções diferentes não significa que elas executem ao mesmo tempo. O desfasamento é apenas uma medida de tempo em relação ao inicio da transacção, o qual pode ocorrer a qualquer altura em relação ao inicio das outras transacções. Usando estes valores, obtiveram-se os resultados da Tabela 5.1, ilustrados na Figura 5.10.

	transacção 1				transacção 2				transacção 3			
alta	*tarefa*	*nó*	*desf.*	*r*	*tarefa*	*nó*	*desf.*	*r*	*tarefa*	*nó*	*desf.*	*r*
p	RS1	1	0	150	RS1	2	0	300	RS1	3	0	600
r	RS2	1	464	150	RS2	2	835	300	RS2	3	1199	600
i	RR1	1	464	300	RR1	1	835	600	RR1	1	1199	1050
o	RR1	2	464	150	RR1	2	835	600	RR1	2	1199	900
r	RR1	3	464	150	RR1	3	835	300	RR1	3	1199	900
i	RR1	>3	464	150	RR1	>3	835	300	RR1	>3	1199	600
d	RR2	1	835	150	RR2	1	1513	600	RR2	1	2177	1050
a	RR2	2	835	150	RR2	2	1513	300	RR2	2	2177	900
d	RR2	3	835	150	RR2	3	1513	300	RR2	3	2177	600
e	RR2	>3	835	150	RR2	>3	1513	300	RR2	>3	2177	600
baixa	*msg*		*desf.*	*r*	*msg*		*desf.*	*r*	*msg*		*desf.*	*r*
	R-DATA		150	314	R-DATA		300	535	R-DATA		600	599
	CONFIRM		614	221	CONFIRM		1135	378	CONFIRM		1799	378

Tabela 5.1: Resultados do cenário complexo

Comparando estes resultados, verifica-se que até a transacção com maior prioridade sofre de alguma interferência, visto que o CAN não interrompe a transmissão de mensagens de menor prioridade. Ao comparar estes valores com os obtidos anteriormente para uma transacção a correr isoladamente ($757\mu s$), a transacção 1 leva 130%($985\mu s$), a transacção 2 leva 279%($2113\mu s$) e a transacção 3 426%($3227\mu s$) mais tempo. Isto denota um crescimento linear do tempo de resposta de cada transacção, desde que o período das transacções seja suficiente de modo a que cada processador não esteja sobrecarregado. Este crescimento linear é de esperar nestes casos, visto que as tarefas de menor prioridade não conseguem interferir com os tempos de resposta das tarefas de menor prioridade. De notar que, embora a transacção 3 exceda a meta usando $D = 3000\mu s$, os resultados obtidos são válidos para $\geq 2777\mu s$.

5.6.7.2 Cenário com efeitos do escalonador, *jitter* e erros na rede

De modo a testar os efeitos do escalonador, o *jitter* e os erros na rede, repetiu-se o cenário anterior mas com os seguintes valores modificados: $D_i = T_{ti=trans(i)} = 5000\mu s$, $T_{clk} = 100\mu s$, $C_{clk} = 20\mu s$, $C_{QL} = 10\mu s$, $C_{QS} = 5\mu s$, $J_i = 50\mu s$ para todas as tarefas e um erro no

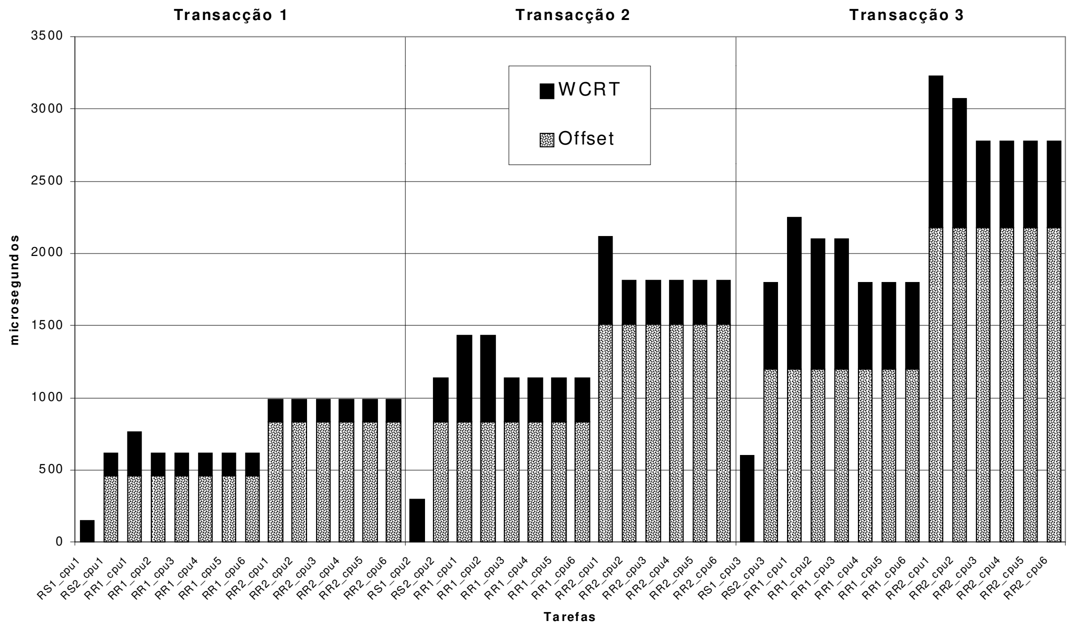

Figura 5.10: Gráfico de valores obtidos no cenário complexo

CAN mínimo, mas que obriga a que qualquer mensagem possa ter de ser retransmitida. Os resultados deste cenário encontram-se na Tabela 5.2 e no gráfico na Figura 5.11.

Como resulta destes cálculos, devido à pouca carga do processador, a computação usada pelo escalonador não afecta o pior tempo de resposta das tarefas. A maioria delas apenas vê o seu pior tempo aumentado do valor do *jitter* em relação aos valores da Tabela 5.1. No entanto, a introdução de erros na rede leva à potencial retransmissão de cada uma das mensagens transmitidas, o que lhes duplica ou mesmo triplica o tempo de resposta. Isto tem consequências no tempo de resposta global de cada transacção que crescem 169%, 168% e 150% respectivamente. A terceira transacção saiu beneficiada neste novo cenário, devido à diminuição do seu tempo de resposta, porque as tarefas que a compõem já não entram tanto em conflito com as da segunda transacção.

	transacção 1				transacção 2				transacção 3			
alta	*tarefa*	*nó*	*desf.*	*r*	*tarefa*	*nó*	*desf.*	*r*	*tarefa*	*nó*	*desf.*	*r*
p	RS1	1	0	200	RS1	2	0	350	RS1	3	0	500
r	RS2	1	828	200	RS2	2	1606	350	RS2	3	2419	500
i	RR1	1	828	350	RR1	1	1606	500	RR1	1	2419	800
o	RR1	2	828	200	RR1	2	1606	500	RR1	2	2419	800
r	RR1	3	828	200	RR1	3	1606	350	RR1	3	2419	800
i	RR1	>3	828	200	RR1	>3	1606	350	RR1	>3	2419	500
d	RR2	1	1470	200	RR2	1	3026	500	RR2	1	4053	800
a	RR2	2	1470	200	RR2	2	3026	350	RR2	2	4053	800
d	RR2	3	1470	200	RR2	3	3026	350	RR2	3	4053	500
e	RR2	>3	1470	200	RR2	>3	3026	350	RR2	>3	4053	500
baixa	*msg*		*desf.*	*r*	*msg*		*desf.*	*r*	*msg*		*desf.*	*r*
	R-DATA		200	628	R-DATA		350	1256	R-DATA		500	1919
	CONFIRM		1026	442	CONFIRM		1954	1070	CONFIRM		2917	1134

Tabela 5.2: Resultados do cenário com efeitos do escalonador, *jitter* e erros no CAN

5.6.7.3 Cenário com alteração das prioridades das mensagens

Finalmente, e para ilustrar como é o impacto da não herança de prioridades de acordo com a prioridade do evento original, construiu-se um cenário em que as mensagens de confirmação eram menos prioritárias do que as de dados. Neste cenário voltou-se a desprezar os efeitos do escalonador, *jitter* e erros no CAN. Os resultados obtidos são apresentados na Tabela 5.3 e no gráfico na Figura 5.12.

A partir dos resultados obtidos, é possível observar que a opção de tornar mais prioritárias as mensagens de dados tem algum impacto negativo no tempo de resposta da primeira transacção, sendo de esperar que a inclusão de transacções extras tenha algum impacto adicional não negligenciável. Por outro lado, nas restantes transacções, é melhorado o tempo de resposta. Em relação ao primeiro cenário, os tempos de resposta de cada transacção são, respectivamente, 122%, 93% e 89%.

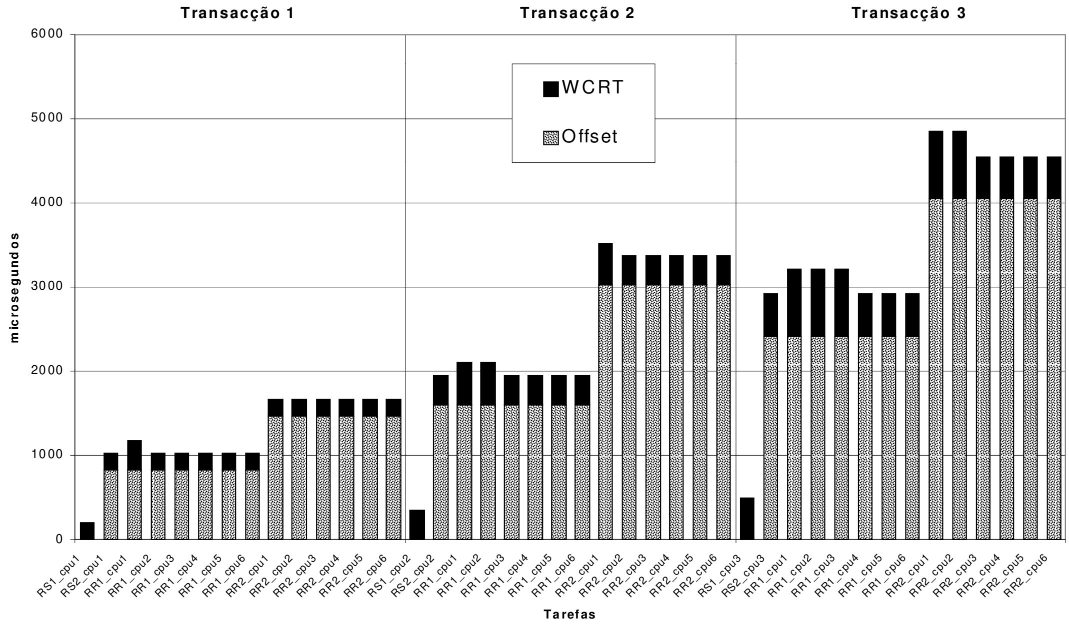

Figura 5.11: Gráfico de valores obtidos no cenário com efeitos do escalonador, *jitter* e erros no CAN

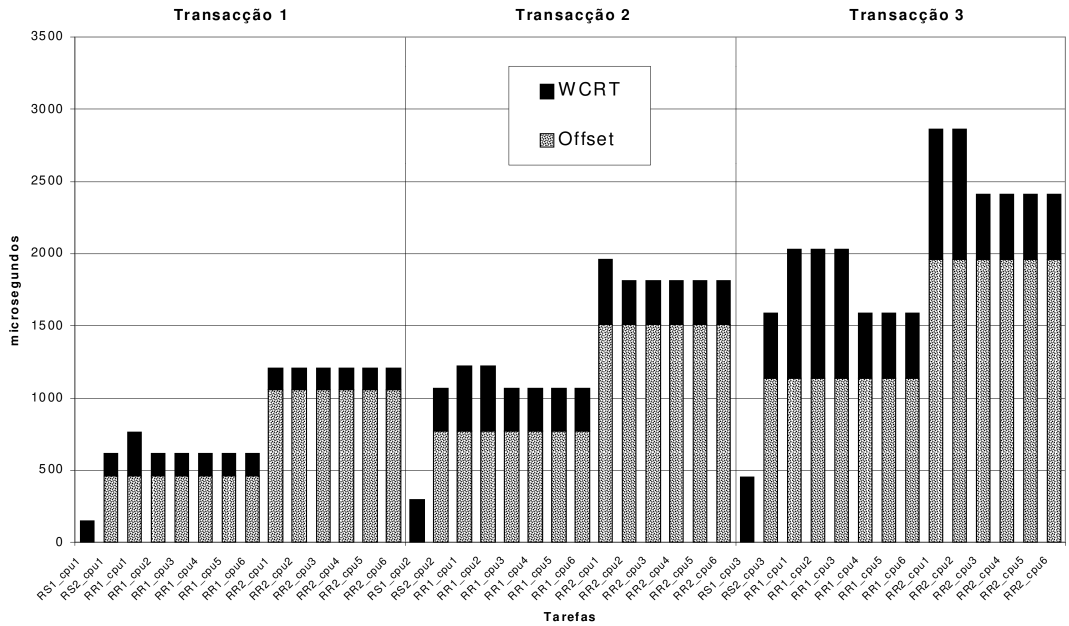

Figura 5.12: Gráfico de valores obtidos no cenário com alteração das prioridades das mensagens

	transacção 1				transacção 2				transacção 3			
alta	*tarefa*	*nó*	*desf.*	*r*	*tarefa*	*nó*	*desf.*	*r*	*tarefa*	*nó*	*desf.*	*r*
p	RS1	1	0	150	RS1	2	0	300	RS1	3	0	450
r	RS2	1	464	150	RS2	2	771	300	RS2	3	1135	450
i	RR1	1	464	300	RR1	1	771	450	RR1	1	1135	900
o	RR1	2	464	150	RR1	2	771	450	RR1	2	1135	900
r	RR1	3	464	150	RR1	3	771	300	RR1	3	1135	900
i	RR1	>3	464	150	RR1	>3	771	300	RR1	>3	1135	450
d	RR2	1	1056	150	RR2	1	1513	450	RR2	1	1963	900
a	RR2	2	1056	150	RR2	2	1513	300	RR2	2	1963	900
d	RR2	3	1056	150	RR2	3	1513	300	RR2	3	1963	450
e	RR2	>3	1056	150	RR2	>3	1513	300	RR2	>3	1963	450
baixa	*msg*		*desf.*	*r*	*msg*		*desf.*	*r*	*msg*		*desf.*	*r*
	R-DATA		150	314	R-DATA		300	471	R-DATA		600	535
	CONFIRM		614	442	CONFIRM		1071	442	CONFIRM		1585	378

Tabela 5.3: Resultados do cenário com alteração das prioridades das mensagens

5.7 Sumário

Neste capítulo, apresenta-se uma aplicação prática da técnica proposta para análise de escalonabilidade a composições de micro-protocolos. Os protocolos escolhidos são um conjunto de protocolos modulares que fornecem serviços de difusão fiável ao barramento CAN. O capítulo começa por descrever o CAN e, em seguida, o seu modelo de rede. Após uma breve descrição da motivação que levou ao desenvolvimento dos protocolos, são apresentados o RELCAN e o EDCAN. Os resultados da aplicação da técnica proposta em vários cenários de utilização foram mostrados e comentados.

Capítulo 6

Conclusões

A aplicação das técnicas de análise de escalonabilidade não é tarefa trivial, sendo necessário um modelo bastante fiel do sistema a estudar de forma a obter resultados o mais fidedignos possível.

Nesta dissertação foi proposta uma metodologia que permite aplicar a técnica de análise de escalonabilidade apresentada de forma a, por um lado obter um valor que majore o pior tempo de resposta, e que por outro, torne estes o menos pessimistas possível.

A técnica de análise apresentada é bastante adequada às especificidades dos sistemas de micro-protocolos, pois permite a utilização das relações de precedência entre as tarefas que fazem parte de uma cadeia de eventos de forma a reduzir o pessimismo.

A metodologia proposta é, na sua configuração actual, destinada aos autores de sistemas de micro-protocolos, que de modo a utilizarem-na terão de realizar manualmente todo o pré-processamento e a construção do ficheiro de entrada para a ferramenta de cálculo do pior tempo de resposta do sistema. Mesmo assim, é já possível usá-la para obter dados relevantes como ficou demonstrado através da aplicação da metodologia ao protocolo RELCAN.

O RELCAN, que assenta sobre o protocolo EDCAN e sobre o barramento CAN, foi originalmente desenvolvido para fornecer serviços de difusão fiável ao CAN. É um exemplo bastante adequado a este estudo, visto que é um micro-protocolo bastante simples, desenhado para sistemas de tempo-real, permitindo por um lado uma fácil compreensão dos resultados obtidos e por outro a aplicação de todos os aspectos da metodologia proposta.

De forma a transformar a metodologia proposta numa ferramenta que permita um uso fácil por parte dos autores de micro-protocolos é necessário desenvolver ainda os seguintes módulos:

- Módulo de extracção de todos os eventos gerados e recebidos, que processaria uma meta-descrição do protocolo, idealmente o próprio código fonte ou então informação gerada por um ambiente integrado de desenvolvimento especialmente adaptado para o efeito. Este módulo deveria ser capaz de eliminar os eventos impossíveis de serem gerados de forma causal a partir de outro.

- Módulo de determinação do pior tempo de computação, que analisaria o código máquina do protocolo e usando um modelo do processador seria capaz de obter o valor pretendido, ou então fazendo parte de uma linguagem própria para codificação de protocolos que imponha as restrições adequadas de forma a permitir obter um máximo para este valor.

Apêndice A

Manual de utilização da ferramenta de análise

A complexidade da técnica de análise de escalonabilidade apresentada no capítulo 3, torna obrigatório o uso de uma ferramenta computacional que a realize. As ferramentas desenvolvidas originalmente durante os trabalhos desta técnica foram disponibilizadas publicamente pelo seu autor.

Para a realização desta dissertação, foi necessário o uso de uma ferramenta que permitisse obter alguns resultados para os cenários desenvolvidos. No entanto, a aplicação das ferramentas já existentes não é trivial, visto que uma delas concretiza uma técnica holística que inclui um modelo de rede específico (acesso TDMA) que não o desejado (CAN) e sem a noção de desfasamentos, enquanto uma outra já inclui os desfasamentos mas com computação num único processador, e consequentemente sem modelo de rede. A possibilidade de desenvolver uma nova ferramenta de raiz chegou a ser equacionada, mas uma análise detalhada de ambas as ferramentas permitiu concluir que seria exequível adicionar as funcionalidades desejadas à ferramenta de análise com desfasamento, evitando assim o desperdício de esforço em duplicar uma parte significativa do código.

A ferramenta original foi então estendida com as seguintes funcionalidades:

- Noção de processamento em processadores distintos;
- Incorporação de uma interface para o modelo de rede;
- Modelo de rede do CAN;
- Alteração do interpretador do ficheiro de configuração de modo a incluir parâmetros para a rede, informação do processador onde cada tarefa é executada e os parâmetros de cada mensagem.

Não sendo pretendido nesta dissertação uma explicação detalhada do desenho da ferramenta, é no entanto útil fornecer uma descrição sucinta do seu funcionamento:

1. Iniciação; Interpretação do ficheiro de configuração;
2. Repetir até todos os valores convergirem:
   - Calcular pior tempo de resposta das tarefas. Actualizar desfasamentos das mensagens para o valor do pior tempo de resposta das tarefas emissoras;
   - Calcular pior tempo de resposta das mensagens. Actualizar desfasamentos das tarefas receptoras para o pior tempo de resposta das respectivas mensagens.
3. Visualizar resultados.

A.1 Argumentos de linha de comando

A ferramenta aceita os seguintes argumentos na sua linha de comandos:

- `-W <nome do ficheiro>` Gera um ficheiro de resultados com os valores separados por vírgulas (formato CSV);
- `-o <nome do ficheiro>` Gera um ficheiro com os resultados, ao invés os visualizar na consola;
- `-i <nome do ficheiro>` Usa o ficheiro dado como ficheiro de configuração;
- `-E` Aplica a análise exacta em vez da análise tratável;
- `-C` Aplica os efeitos de escalonamento;
- `-w <número>` Tamanho da janela de medição dos efeitos de escalonamento;
- `-S <número>` Tempo de duração da simulação do sistema;
- `-I` Calcular valor do maior período ocupado;
- `-p[r|R|D|T|e|B|C|O|J|s|o|t|c]` Resultados a serem visualizados:
  - `r` Pior tempo de resposta;
  - `R` Pior tempo de resposta da análise original (sem desfasamentos);
  - `D` Meta;
  - `T` Período da transacção;
  - `e` Parâmetro e (de quantas em quantas transacções é que a tarefa é activada);
  - `B` Factor de bloqueamento;
  - `C` Pior tempo de computação;
  - `O` Desfasamento;
  - `J` *Jitter*;
  - `s` Informação de semáforos;

- `o` Informação de objectos;
- `t` Informação de tarefas/transacções;
- `c` Informação de conformidade com as restrições de precedência e exclusão;

A.2 Ficheiro de configuração

A prioridade das tarefas e mensagens é dada de acordo com a sua ordem no ficheiro de configuração. À excepção dos parâmetros de escalonamento, todas as seguintes linhas podem aparecer mais do que uma vez.

Parâmetros de escalonamento

`parameters` *<Tclk> <Cclk> <CQL> <CQS> <Máx. Tempo Resposta>*

Esta linha é usada para atribuir valores aos parâmetros de escalonamento, mais detalhes ver secção 3.1.5.

Processadores

`processor` *<identificador>*

Esta linha declara um processador no sistema, sendo atribuído a este o identificador dado.

Redes

`network` *<identificador> <tipo>*

Esta linha declara uma rede do tipo fornecido com o identificador dado.

Transacções

`trans` *<identificador> <período>*

Declara a transacção com o identificador e o período dados.

Tarefas

`task` *<identificador> <transacção> <processador> <e> <Meta> <C> <J> <Desfasamento>*

Declara uma tarefa com o identificador dado, pertencente à transacção e executando no processador com os parâmetros dados.

Mensagens

`message` *<identificador <transacção> <rede> <e> <Meta> <C> <J> <emissor> <receptor1> ... <receptorN>*

Esta linha declara uma mensagem identificada pelo identificador dado, que é enviada na rede com os parâmetros fornecidos. A tarefa emissora e as várias tarefas receptoras são também explicitadas.

Semáforos

`sem` *<identificador>*

Declara um semáforo com o identificador fornecido.

Acesso a estruturas de exclusão mútua

`lock` *<semáforo> by <tarefa> <tempo>*

Declara a possibilidade da tarefa dada reservar o semáforo para si durante o tempo dado.

Precedências

`precedence` *<tarefa1> <tarefa2>*

Declara uma relação de precedência entre duas tarefas. A segunda tarefa só pode começar a executar depois da primeira terminar.

Exclusão

`exclusion` *<tarefa1> <tarefa2>*

Declara as tarefas indicadas como sendo de execução exclusiva, i.e. uma das duas tem de terminar antes da outra poder executar.

Bibliografia

[1] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, Department of Computer Science, University of York, York, YO10 5DD, UK, Dec. 1991.

[2] N. C. Audsley, A. Burns, M. F. Richardson, K. W. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, Sept. 1993.

[3] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard real-time scheduling: The deadline monotonic approach. In *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atlanta, GA, USA, May 1991.

[4] N. T. Bhatti, M. A. Hiltunen, R. D. Schlichting, and W. Chiu. Coyote: A system for constructing fine-grain configurable communication services. *ACM Transactions on Computer Systems*, 16(4):321–366, Nov. 1998.

[5] K. P. Birman and R. van Renesse. *Reliable Distributed Computing with the ISIS Toolkit*. IEEE Computer Society Press, 1994.

[6] G. Bollella, B. Brosgol, P. Dibble, S. Furr, J. Gosling, D. Hardin, and M. Turnbull. *The Real-Time Specification for Java*. Addison-Wesley, preliminary edition, June 2000. ISBN 0-201-70323-8.

[7] A. Burns and A. J. Wellings. *Real-Time Systems and Programming Languages*, chapter 13. International Computer Science Series. Addison-Wesley, 2nd edition, 1996. ISBN 0-201-40365-X.

[8] J. C. P. Gutiérrez, J. J. G. Garcia, and M. G. Harbour. Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems. In *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*, pages 35–44, Berlin, Germany, June 1998.

[9] M. Hayden. The ensemble system. Technical Report TR 98-1662, Cornell University, Ithaca, NY, USA, 1998.

[10] M. A. Hiltunen, R. D. Schlichting, X. Han, M. M. Cardozo, and R. Das. Real-time dependable channels: Customizing QoS attributes for distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):600–612, June 1999.

[11] N. Hutchinson and L. L. Peterson. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, Jan. 1991.

[12] ISO. *ISO International Standard 7498 - Information processing systems - Open Systems Interconnection - Basic Reference Model*. International Standards Organization, 1984.

[13] ISO. *ISO International Standard 11898 - Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for high-speed communication*. International Standards Organization, Nov. 1993.

[14] M. Joseph and P. K. Pandya. Finding response times in a real-time system. *BCS Computer Journal*, 29(5):390–395, Oct. 1986.

[15] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer International Series in Engineering and Computer Science. Real-Time Systems. Kluwer Academic Publishers, 1997. ISBN 0-7923-9894-7.

[16] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, M. Schwabl, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: The MARS approach. *IEEE Micro*, 9(1):25–40, 1989.

[17] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pages 201–209, Lake Buena Vista, FL, USA, Dec. 1990.

[18] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterisation and average case behaviour. In *Proceedings of the 10th IEEE Real-Time Systems Symposium*, pages 166–171, Santa Monica, CA, USA, Dec. 1989.

[19] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, Jan. 1973.

[20] H. Miranda, A. Pinto, and L. Rodrigues. Appia, a flexible protocol kernel supporting multiple coordinated channels. In *Proceedings of the 21st International Conference on Distributed Computing Systems*, Phoenix, AZ, USA, Apr. 2001.

[21] J. C. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 26–37, Madrid, Spain, Dec. 1998.

[22] J. C. Palencia and M. G. Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 328–339, Phoenix, AZ, USA, Dec. 1999.

[23] L. M. Pinho, F. Vasques, and E. Tovar. Integrating inaccessibility in response time analysis of CAN networks. In *Proceedings of 3rd IEEE International Workshop on Factory Communication Systems*, pages 77–84, Porto, Portugal, Sept. 2000.

[24] J. Postel. RFC793: Transmission control protocol, DARPA internet program, protocol specification, Sept. 1981.

[25] S. Punnekkat, H. Hansson, and C. Norström. Integrating reliability and timing analysis of CAN-based systems. In *Proceedings of 3rd IEEE International Workshop on Factory Communication Systems*, pages 165–172, Porto, Portugal, Sept. 2000.

[26] S. Punnekkat, H. Hansson, and C. Norström. Response time analysis under errors for CAN. In *Proceedings of 6th IEEE Real Time Technology and Applications Symposium*, Washington D.C., USA, June 2000.

[27] P. Puschner and A. Burns. A review of worst-case execution-time analysis (editorial). *Real-Time Systems*, 18(2/3):115–128, May 2000.

[28] K. Ramamritham and J. A. Stankovic. Dynamic task scheduling in hard real-time distributed systems. *IEEE Software*, 1(3):65–75, July 1984.

[29] J. Rodrigues, H. Miranda, J. Ventura, and L. Rodrigues. The design of RT-Appia. In *Proceedings of the 6th International Workshop on Object-Oriented Real-Time Dependable Systems*, Rome, Italy, Jan. 2001.

[30] J. Rufino. An overview of the Controller Area Network. In *Proceedings of the CiA Forum CAN for Newcomers*, Braga, Portugal, Jan. 1997.

[31] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcasts in CAN. In *Digest of Papers, The 28th IEEE International Symposium on Fault-Tolerant Computing*, pages 150–159, Munich, Germany, June 1998.

[32] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, Sept. 1990.

[33] K. W. Tindell. Using offset information to analyse static priority pre-emptively scheduled task sets. Technical Report YCS 182, Department of Computer Science, University of York, York, YO10 5DD, UK, Sept. 1992.

[34] K. W. Tindell. Adding time-offsets to schedulability analysis. Technical Report YCS 221, Department of Computer Science, University of York, York, YO10 5DD, UK, Jan. 1994.

[35] K. W. Tindell and A. Burns. Guaranteed message latencies for distributed safety-critical hard real-time control networks. Technical Report YCS 229, Department of Computer Science, University of York, York, YO10 5DD, UK, May 1994.

[36] K. W. Tindell, A. Burns, and A. J. Wellings. Calculating controller area network (CAN) message response times. In *Proceedings of the IFAC Workshop on Distributed Computer Control Systems*, pages 35–40, Toledo, Spain, Sept. 1994.

[37] K. W. Tindell, A. Burns, and A. J. Wellings. An extendible approach for analysing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, Mar. 1994.

[38] K. W. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing & Microprogramming*, 40(2-3):117–134, Apr. 1994.

[39] F. Travostino, E. F. Menze, and F. D. Reynolds. Paths: Programming with system resources in support of real-time distributed applications. In *Proceedings of the 2nd International Workshop on Object-Oriented Real-Time Dependable Systems*, Laguna Beach, CA, USA, Feb. 1996.

[40] R. van Renesse, K. P. Birman, and S. Maffeis. Horus: A flexible group communications system. *Communications of the ACM*, 39(4):76–83, Apr. 1996.

[41] P. Veríssimo, P. Barrett, P. Bond, A. Hilborne, L. Rodrigues, and D. Seaton. *Delta-4 – A Generic Architecture for Dependable Distributed Computing*, chapter The Extra Performance Architecture (XPA), pages 211–266. Springer Verlag, 1991.

[42] P. Veríssimo and H. Kopetz. *Distributed Systems*, chapter 19. ACM Press Frontier. Addison-Wesley, 2nd edition, 1995. ISBN 0-201-62427-3.