Universidade Federal da Bahia
Instituto de Matemática / Escola Politécnica
Departamento de Ciência da Computação / Departamento de Engenharia Mecânica

Programa de Pós Graduação em Mecatrônica

Verificação Formal da Função de Controle de Acesso ao Meio do Protocolo IEEE 802.11 e Investigação da sua Aplicabilidade em Sistemas de Tempo-Real

Frederico Jorge Ribeiro Barboza

Dissertação de Mestrado
Salvador
26 de junho de 2006

Universidade Federal da Bahia
Instituto de Matemática / Escola Politécnica
Departamento de Ciência da Computação / Departamento de Engenharia Mecânica

Frederico Jorge Ribeiro Barboza

Verificação Formal da Função de Controle de Acesso ao Meio do Protocolo IEEE 802.11 e Investigação da sua Aplicabilidade em Sistemas de Tempo-Real

Trabalho apresentado ao Programa de Programa de Pós Graduação em Mecatrônica do Departamento de Ciência da Computação / Departamento de Engenharia Mecânica da Universidade Federal da Bahia como requisito parcial para obtenção do grau de Mestre em Mecatrônica.

Orientadora: *Profa. Dra. Aline Maria dos Santos Andrade*
Co-orientador: *Prof. Dr. Flávio Moraes Assis Silva*
Co-orientador: *Prof. Dr. George Marconi de Araújo Lima*

Salvador
26 de junho de 2006

Ficha Catalográfica elaborada pela Biblioteca Bernadete Sinay Neves, Escola Politécnica da UFBA.

Barboza, Frederico Jorge Ribeiro

B239p Verificação formal da função de controle de acesso ao meio do protocolo IEEE 802.11 e investigação da sua aplicabilidade em sistemas de tempo-real / Frederico Jorge Ribeiro Barboza. - Salvador, 2006.

111 f. : il.

Orientadora: Profa. Dra. Aline Maria dos Santos Andrade.

Co-orientadores: Prof. Dr. Flávio Moraes Assis Silva e Prof. Dr. George Marconi de Araújo Lima.

Dissertação (mestrado) - Universidade Federal da Bahia, Escola Politécnica, Instituto de Matemática, 2005.

1. Programação em tempo real. 2. Sistemas de comunicação sem fio. I Andrade, Aline Maria dos Santos. II. Silva, Flávio Moraes Assis. III. Lima, George Marconi de Araújo. IV. Universidade Federal da Bahia. Escola Politécnica. V. Universidade Federal da Bahia. Instituto de Matemática. VI. Título.

CDD 20.ed. 004.3

TERMO DE APROVAÇÃO

FREDERICO JORGE RIBEIRO BARBOZA

VERIFICAÇÃO FORMAL DA FUNÇÃO DE CONTROLE DE ACESSO AO MEIO DO PROTOCOLO IEEE 802.11 E INVESTIGAÇÃO DA SUA APLICABILIDADE EM SISTEMAS DE TEMPO-REAL

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre em Mecatrônica, Universidade Federal da Bahia, pela seguinte banca examinadora:

Aline Maria Santos Andrade - Orientadora _____

Doutora em Informática. Pontifícia Universidade Católica do Rio de Janeiro (PUC-RJ)

Universidade Federal da Bahia

Eduard Montgomery Meira Costa _____

Doutor em Engenharia Elétrica. Universidade Federal da Paraíba (UFPB)

Universidade Federal da Bahia

Edward Hermann Haeusler _____

Doutor em Informática. Pontifícia Universidade Católica do Rio de Janeiro (PUC-RJ)

Pontifícia Universidade Católica do Rio de Janeiro

José Augusto Suruagy Monteiro _____

Doutor em Ciência da Computação. University Of California Los Angeles (U.C.L.A.), Estados Unidos

Universidade Salvador

Salvador, 26 de junho de 2006

A Adri e Luca, por fazerem as coisas sempre fazerem sentido.

Agradecimentos

Ainda que uma lista de agradecimentos sempre corra o risco de ser injusta, por ser incompleta, devo assumir este risco por não poder deixar de agradecer:

Aos meus pais e as minhas mães, por serem os principais responsáveis pelo que sei e, principalmente, pelo que sou e às minhas irmãs, por partilharem comigo os caminhos;

A Adri, pela eterna paciência e suporte;

Aos meus orientadores, Aline, Flávio e George, sempre dispostos ao auxílio, extrapolando as atribuições de mera orientação, verdadeiros amigos e incentivadores;

Aos amigos e professores do LaSiD, em especial a Macêdo, por sonhar e trabalhar arduamente na construção dos sonhos;

Aos demais professores do DCC da UFBa, em particular, a Débora, Fabíola e Cristina, por acreditarem em um curso melhor e por lutarem por esta crença;

Aos grandes amigos que fiz ainda na graduação, pessoas que me acompanham a uma década e que fizeram aquele período valer a pena: Niv, Cog, Deco, Get, Zasc e Igaus;

Aos amigos da primeira turma do Mestrado Acadêmico em Mecatrônica, por terem compartilhado as certezas, as dúvidas e as descobertas, em especial, a Sandro, Ivo, Rodrigo, Jair, Márcio e Luciano;

Aos amigos da Faculdade Ruy Barbosa, pelo permanente apoio, especialmente, a Ana Pat, Angel, Manoelzinho, Mônica, Marcela e Simone;

Aos professores do PPGM pelas lições de vida, muito além da simples informação;

Aos Professores Eduard, Hermann e Suruagy, membros da banca examinadora, pela imensurável colaboração na revisão do texto final e pelas críticas e sugestões apresentadas para o aprimoramento deste trabalho;

Ao Tribunal Regional Eleitoral da Bahia pela licença capacitação concedida, prova inconteste da crença na necessidade de capacitação dos seus servidores;

Às crianças, Luca e Júlia, por ‘compreenderem’ os momentos de ‘distância’;

Aos meus amigos, sempre presentes, mesmo quando, por diversas vezes, estive ausente: Manoel, Carol, Hélio, Mônica, Lu, Beto, Lulú, China, Moniquinha, Joana, Mateus, Gui, Joel, Gessé, Chris, Lela, Lívia, Dani, Dado, Su, Rose, Kari, Carla, Murilo, Alex, Villa, Cabeça, Jú, Léo, Deca, ..., ..., ...

A coisa principal da vida não é o conhecimento, mas o uso que dele se faz.
— (Talmude)

Resumo

O termo IEEE 802.11 diz respeito a uma família de especificações que buscam obter conectividade sem fio para estações fixas, portáveis e móveis em uma rede local. Redes IEEE 802.11 têm, recentemente, despertado interesse como tecnologia de suporte para a comunicação em aplicações sem fio na automação, em particular em aplicações de chão de fábrica e de controle de plantas, onde, muitas vezes, requisitos de tempo-real e requisitos de confiabilidade são necessários.

Neste contexto, o uso de métodos formais permite a obtenção de um conhecimento mais preciso sobre as propriedades do protocolo bem como a especificação e verificação destas propriedades.

Este trabalho apresenta uma especificação e verificação formal da função de controle de acesso ao meio da sub-camada MAC do padrão IEEE 802.11 usando UPPAAL, um verificador de modelos gratuito, que suporta os conceitos de relógios e tempo. O uso do UPPAAL permitiu considerar, dentro da modelagem, as características temporais do protocolo.

A verificação procurou identificar uma série de propriedades que fornecesse aos projetistas de aplicações e sistemas de tempo-real um conjunto mínimo de garantias relativas aos canais de comunicação. Entre as propriedades verificadas, destacamos a habilidade das estações possuírem acesso ao meio dentro de um tempo finito e conhecido e, portanto, a adequação do protocolo como suporte a aplicações que necessitem de garantias temporais.

Palavras-chave: Métodos Formais; Sistemas de Tempo-Real; Confiabilidade de Sistemas; Especificação de Sistemas; Engenharia de Software.

Abstract

IEEE 802.11 denotes a set of Wireless LAN/WLAN standards developed by working group 11 of the IEEE LAN/MAN Standards Committee (IEEE 802) for wireless connectivity for fixed, portable, and moving stations within a local area.

In the past few years protocols based on the IEEE 802.11 specification and applications that make use of it have become increasingly popular. More recently, this wireless network standard has been required to support systems that need quality-of-service (QoS) and/or real-time guarantees.

This work presents a formal specification and verification of the IEEE 802.11 standard, where the PCF and DCF functions are considered in an integrated way and timing properties were taken into account. To do so, we made use of UPPAAL, a model checking tool. UPPAAL provides the concepts of time and clocks, allowing for the modelling, simulation and verification of computing systems that require timing guarantees.

Using UPPAAL, we have represented the temporal behavior of the IEEE 802.11 standard and modelled the temporal effects of the integration between the PCF and DCF functions. Also, we formally proved important properties of the IEEE 802.11 specification. Among the verified properties we emphasize the ability of workstations to have medium access within a bounded known time and so the ability of the IEEE 802.11 to support applications that require timing guarantees.

Keywords: Formal methods; Real-time systems; Software reliability; Software specification, Software engineering.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ACK — Acknowledgment
AP — Access Point
ATIM — Announcement Traffic Indication Message
BSA — Basic Service Area
BSS — Basic Service Set
CCA — Clear Channel Assessment Function
CF — Coordination Function
CFP — Contention Free Period
CFPRate — Contention Free Repetition Rate
CP — Contention Period
CRC — Cyclic Redundancy Code
CSMA/CA — Carrier Sense Multiple Access / Collision Avoidance
CTS — Clear To Send
CW — Contention Window
DCF — Distributed Coordination Function
DIFS — DCF Interframe Spacing
DS — Distribution System
DSSS — Direct Sequence Spread Spectrum
EIFS — Extended Interframe Spacing
ESS — Extended Service Set
FCS — Frame Check Sequence
FHSS — Frequency Hopping Spread Spectrum
IBSS — Independent Basic Service Set
IEEE — Institute of Electrical and Electronics Engineers
IFS — Interframe Spacing
IR — Infra-Red
LAN — Local Area Network
MAC — Medium Access Control
MAN — Metropolitan Area Network
MPDU — MAC Protocol Data Unit
MSDU — MAC Service Data Unit
NAV — Network Allocation Vector
PAN — Personal Area Network
PC — Point Coordination
PCF — Point Coordination Function

PHY — Physical Layer
PIFS — PCF Interframe Space
RM-OSI/ISO — Reference Model - Open Systems Interconnection / International Organization for Standardization
RTS — Request To Send
SIFS — Short Interframe Spacing
WLAN — Wireless Local Area Network
WPAN — Wireless Personal Area Network

CAPÍTULO 1

Introdução

Só sabemos com exatidão quando sabemos pouco; à medida que vamos adquirindo conhecimentos, instala-se a dúvida.

—JOHANN GOETHE (Filósofo Alemão)

Sistemas Mecatrônicos de Tempo-Real

Cada vez mais sistemas automáticos vêm sendo utilizados em aplicações de natureza crítica. É comum encontrarmos máquinas realizando o controle do tráfego aéreo, de plantas nucleares, de sistemas de defesa e de plantas industriais. O uso da automação no controle deste tipo de aplicação fornece diversos benefícios ao homem. Dentre estes benefícios podemos destacar a preservação da exposição humana a ambientes insalubres ou de riscos, como é o caso das aplicações de controle de plantas nucleares, prospecção de vulcões e águas profundas e de exploração espacial; a liberação da necessidade de execução de tarefas repetitivas, como é o caso dos sistemas de controle em geral; e a necessidade de precisão na execução de determinadas atividades, como é o caso do controle de tráfego aéreo e dos sistemas embarcados de navegação.

Em comum, os sistemas descritos possuem como característica o fato de necessitarem interagir massivamente com o meio no qual encontram-se imersos, através da aquisição de informações sobre o estado atual da aplicação e, se necessário, através da modificação deste estado para uma situação desejada. Tais sistemas são tipicamente mecatrônicos. Sistemas mecatrônicos são sistemas que buscam combinar de modo sinérgico as características dos sistemas mecânicos e eletrônicos, primordialmente para a aquisição de sinais como entrada e a geração de força e movimento como saída, com a necessidade de processamento dos sinais obtidos através de um sistema computacional de controle (SK97). Assim, a adoção de sistemas mecatrônicos vem sendo de grande utilidade em diversos cenários. Deve-se observar, entretanto, que a ocorrência de falhas nestes sistemas pode implicar em sérios riscos à vida humana, ao meio-ambiente e à economia.

Em diversas situações, a correta funcionalidade de tais sistemas está associada não apenas ao resultado que eles produzem, mas também ao momento em que este resultado é produzido. Por exemplo, um sistema de controle de sinais vitais em uma Unidade de Terapia Intensiva deve fazer soar um alarme ou atuar de uma determinada forma sobre o paciente em um certo tempo previsto, a partir da mudança da situação do paciente observado. Caso o sistema não atue no tempo desejado, a saúde ou até mesmo a vida do paciente poderá estar invariavelmente comprometida. Sistemas mecatrônicos que possuem estas características são chamados de sistemas mecatrônicos de tempo-real (Sta93).

Sistemas de tempo-real são particularmente difíceis de especificar e projetar, pois, além das dificuldades inerente às funcionalidades envolvidas, ainda devem levar em conta aspectos do comportamento temporal. Esta característica torna os sistemas de tempo-real ainda mais sujeitos a falhas de projeto e construção. Adicionalmente, em diversos casos, a natureza distribuída da aplicação de tempo-real e a necessidade de introduzir mecanismos de tolerância a falha, de forma a aumentar a disponibilidade destes sistemas, determinam a necessidade de tais sistemas serem dotados de concorrência e paralelismo.

Entretanto, sistemas de tempo-real distribuídos e concorrentes são ainda mais difíceis de projetar que sistemas de tempo-real seqüenciais e centralizados. Isto ocorre principalmente em função das diversas formas distintas de interação entre as partes componentes do sistema, da natureza complexa das interações existentes entre estas partes e o ambiente no qual elas atuam, da inexistência de uma referência única de tempo, típica dos sistemas distribuídos e, finalmente, da necessidade de se considerar e prever o comportamento associado aos canais de comunicação.

Redes sem Fio

Em diversas situações, os requisitos da aplicação evidenciam a necessidade da conectividade entre as partes componentes do sistema ser realizada através de canais sem fio. As redes sem fio fornecem uma alternativa de conectividade em locais onde o uso das redes tradicionais baseadas em cabo torna-se inadequado, como é o caso de prédios históricos e tombados, contruções em concreto ou prédios de amplos vãos livres, como os armazéns e supermercados. Em outras situações, o próprio ambiente operacional impede a implantação de uma estrutura tradicional de rede como nas operações militares, de defesa, salvamento, quando os agentes computacionais estão embarcados em dispositivos robóticos móveis, ou ainda quando a sua necessidade é transitória e torna esta implantação economicamente inviável, como é o caso das pesquisas em campo.

Adicionalmente, uma grande prática que vem sendo sistematicamente adotada é a utilização de componentes de prateleira, ou seja, componentes padronizados disponíveis para venda e utilização do público em geral, no projeto e construção dos sistemas de tempo-real (LH02, Ram99). Com o uso de componentes de prateleira padronizados, os projetistas de sistemas de tempo-real buscam satisfazer uma série de requisitos do mercado, tais como:

- a redução do tempo de desenvolvimento dos sistemas de tempo-real através da reutilização de projetos e soluções;
- o aumento da confiabilidade dos sistemas, devido à utilização de componentes testados e potencialmente contruídos por fornecedores com maior conhecimento na área particular de cada um dos componentes;
- diminuição dos custos de projeto através da utilização de componentes produzidos em larga escala;
- melhor gerenciamento da complexidade dos sistemas, através da divisão desta complexidade em pequenos componentes autônomos de propriedades conhecidas;
- aumento da interoperabilidade dos sistemas, devido ao uso de componentes com interfaces padronizadas.

Neste cenário, o uso de redes comerciais como meio de comunicação para os sistemas de tempo-real distribuídos passou a assumir considerável importância (CV94, TV98). Em particular, merece especial atenção os padrões comerciais para redes de comunicação sem fio.

Dentre os padrões comerciais de rede sem fio, deve-se destacar o padrão IEEE 802.11 (IEE99) devido à importância prática e comercial que este padrão vem assumindo. A IEEE 802.11 corresponde a uma especificação relacionada à subcamada de acesso ao meio (MAC — *Medium Access Control*) e à camada física do modelo de referência RM-OSI/ISO (*Reference Model — Open Systems Interconnection / International Organization for Standardization*), que permite a conexão sem fio de estações de uma rede local. Redes baseadas neste padrão já vêm sendo adotadas na indústria e, nos últimos anos, passaram a merecer especial atenção como meio de suporte a sistemas que requeiram garantias temporais (SGZ01, Zha03, YWB01, LL01).

Como tais aplicações requerem um elevado nível de confiabilidade de comunicação, faz-se necessário obter não apenas um entendimento preciso das propriedades oferecidas pelo padrão, mas também garantias de correção do mesmo. Fornecer tais garantias para um protocolo com o nível de complexidade do IEEE 802.11 é um desafio. Neste contexto, métodos formais podem ser utilizados como ferramenta de auxílio.

Métodos Formais

Para tratar da complexidade e fornecer garantias da correção, os métodos formais podem ser utilizados como uma ferramenta de auxílio ao projeto de sistemas de tempo-real. Métodos formais compõem o conjunto de linguagens, técnicas e ferramentas baseadas em modelos matemáticos e destinadas a fornecer um padrão rigoroso de modelagem e análise de sistemas computacionais. Muitas destas ferramentas têm sido construídas através da utilização de uma técnica conhecida por verificação de modelos (CGP00). A verificação de modelos é baseada na exploração do espaço de estados do sistema analisado, com vistas a identificar se tal sistema satisfaz uma determinada propriedade, ou, caso contrário, fornecer contra-exemplos de execução do sistema que viole esta propriedade. Nesta abordagem, propriedades são normalmente expressas em uma lógica proposicional temporal e os sistemas são modelados como sistemas de transição de estados finitos. Um algoritmo de busca é implementado de forma a determinar se a especificação é verdadeira neste sistema de transição. A abordagem de verificação de modelos merece destaque devido ao fato de analisar o sistema de forma automática, fornecendo resultados em um tempo aceitável em muitos casos.

Adicionalmente, os verificadores de modelos podem ser utilizados para verificar especificações parciais, e, portanto, fornecer informações úteis sobre a correção de um sistema que ainda não esteja completamente especificado. Outra grande vantagem dos verificadores de modelo é o fornecimento de contra-exemplos quando uma determinada propriedade não é satisfeita, auxiliando o projetista na correção de erros.

Contribuições do Trabalho

Este trabalho apresenta uma especificação e verificação formal das funções de controle de acesso ao meio do padrão IEEE 802.11, considerando os seus modos PCF (*Point Coordination Function*) e DCF (*Distributed Coordination Function*) de forma integrada. Para este fim, fez-se uso do UPPAAL (BLL$^+$95), uma ferramenta desenvolvida conjuntamente pelo BRICS (*Basic Research In Computer Science*) da *Aalborg University* da Dinamarca e pelo *Department of Computer Systems* da *Uppsala University* da Suécia. O UPPAAL suporta conceitos de tempo, permitindo a modelagem, simulação e verificação de sistemas computacionais de tempo-real. Esta ferramenta vem sendo utilizada com sucesso para a prova de correção de uma larga gama de aplicações da indústria (BGK$^+$96, HL97, HLS99), incluindo alguns protocolos de comunicação (LP97).

Na especificação do protocolo IEEE 802.11, realizado no modelo de especificação da ferramenta, foi possível se representar o comportamento temporal do padrão. Em seguida, foram

realizadas verificações de uma série de propriedades resultantes da alternância temporal entre as duas funções de coordenação, centralizada e distribuída.

Como principal contribuição, este trabalho apresenta uma demonstração automática de propriedades que evidenciam a adequação das redes IEEE 802.11 como suporte de comunicação para aplicações que possuam restrições temporais. Estas propriedades não haviam sido até então formalmente demonstradas, quando considerado o comportamento integrado das funções de coordenação do protocolo.

Deve-se destacar que, embora o padrão esteja relativamente bem estabelecido, equipamentos de rede que suportem a PCF não vêm sendo construídos comercialmente. Este fato torna ainda mais relevante os resultados apresentados, na medida que considera as propriedades de uma especificação que ainda não foi largamente implementada, permitindo, portanto, o conhecimento do comportamento dos dispositivos com um alto grau de confiabilidade.

Como segunda contribuição, este trabalho fornece um estudo de caso do uso de um verificador de modelos na verificação de propriedades funcionais e temporais de um protocolo real para ambientes sem fio. Com isto, busca-se consolidar a confiança da indústria no uso de métodos formais para o desenvolvimento de sistemas confiáveis.

Estrutura do Texto

Este texto está organizado da seguinte forma:

- o capítulo 2 apresenta os conceitos básicos relativos aos sistemas de tempo-real distribuídos e sem fio e aos métodos formais e faz considerações sobre trabalhos relacionados;
- o capítulo 3 descreve a arquitetura das redes padrão IEEE 802.11 e como ocorre o controle de acesso ao meio nestas redes;
- o capítulo 4 descreve as linguagens de modelagem e especificação utilizadas no verificador de modelo UPPAAL;
- o capítulo 5 apresenta a especificação formal dos elementos que compõem as funções de acesso ao meio da camada MAC do protocolo IEEE 802.11;
- o capítulo 6 relaciona as propriedades submetidas ao verificador e descreve os resultados obtidos;
- finalmente, o capítulo 7 apresenta as conclusões relativas a este trabalho e indica alguns possíveis trabalhos futuros.

CAPÍTULO 2

Sistemas Mecatrônicos de Tempo Real e Sem Fio e Aspectos de Confiabilidade

O termo 'todo' é sempre comparativo.

—IMMANUEL KANT (Filósofo Alemão)

Recentemente, houve um aumento na ploriferação de computadores pessoais e outros dispositivos computacionais portáteis e móveis. Adicionalmente, persistiu a necessidade de se fornecer conectividade e acesso a serviços de redes, independente desta mobilidade. Desta forma, como a necessidade do uso de cabos para conexão dos dispositivos computacionais reduz absurdamente a possibilidade de mobilidade dos seus usuários, tecnologias que fornecessem serviços similares àqueles fornecidos pelas redes com fio, sem as limitações de deslocamento imposta por este tipo de rede, passaram a despertar grande interresse. Isto possibilitou o desenvolvimento de tecnologias de conectividade sem fio, tais como as redes locais sem fio (LANs — Local Area Networks) e redes pessoais sem fio (PANs — Personal Area Networks). Com a popularização destas tecnologias, as redes sem fio passaram a ser comumente encontradas em ambientes residenciais e de escritório.

Após se disseminar no ambiente comercial, a computação sem fio vem, atualmente, aumentando sua presença dentro do ambiente industrial. Este aumento não se dá apenas em termos de aplicações de supervisão, mas também nas aplicações de chão de fábrica e de controle das plantas. É neste contexto que encontra-se inserido o desenvolvimento de sistemas mecatrônicos de tempo-real distribuídos e sem fio.

A utilização de padrões de mercados no projeto dos sistemas mecatrônicos de tempo-real distribuídos e sem fio vêm sendo cada vez mais investigada. Esta prática traz uma série de vantagens quando comparada ao desenvolvimento de projetos de modo *ad-hoc* e com a utilização de componentes proprietários, tais como a redução de custo e a divisão da complexidade do projeto. Em particular, busca-se, em diversos projetos, a adoção de redes padronizadas como suporte de comunicação para estes sistemas.

Entretanto, os sistemas mecatrônicos de tempo-real distribuídos e sem fio requerem muitas vezes um alto grau de confiabilidade e, portanto, demandam técnicas mais rígidas no seu projeto e construção. Deste modo, para aumentar a confiabilidade dos sistemas em construção, é desejável que seus componentes sejam bem investigados e forneçam ao projetista um conjunto de propriedades bem definidas. Uma alternativa que se mostra adequada tanto para o conhecimento detalhado dos sub-sistemas, de forma a reduzir a ambigüidade e futuros erros, como para verificar se a modelagem do sistema atende as propriedades desejadas, é o uso de métodos formais de desenvolvimento de software.

Neste capítulo é apresentada na seção 2.1 a tecnologia de conectividade sem fio e os principais padrões de mercado que suportam esta tecnologia e é discutida a adoção destas redes na automação. Na seção 2.2, apresenta-se a definição de sistemas de tempo-real, explicita-se a necessidade de se utilizar conectividade sem fio nestes sistemas e de como a confiabilidade neste tipo de sistemas está relacionada à previsibilidade. A seção 2.3 apresenta como o uso de métodos formais pode fornecer previsibilidade, e conseqüente confiabilidade, no projeto de sistemas de computação, em particular nos sistemas de tempo-real. Finalmente, a seção 2.4 apresenta alguns trabalhos desenvolvidos com o objetivo de se investigar formalmente as propriedades das redes sem fio, em particular do padrão IEEE 802.11.

2.1 Redes sem Fios

As redes sem fio constituem-se como uma alternativa tecnológica a suas contra-partes com fios. Estas redes oferecem uma série de vantagens quando comparadas às redes com cabeamento tradicionais, dentre os quais podemos destacar a facilidade em adicionar, excluir e modificar as configurações da rede sem os altos custos com passagem de cabos e modificações de conexões envolvidos na alteração das configurações das redes convencionais. Em determinadas situações como, por exemplo, na montagem de redes em construções que tornam difícil ou que impossibilitam a instalação de cabos, como é o caso de prédios tombados como patrimônios históricos, construções em concreto e alvenaria ou galpões com vãos de grandes dimensões, as redes sem fio podem se configurar como a única alternativa possível. Outras situações em que o uso de redes sem fio se constituem uma opção mais viável que as redes tradicionais incluem: aplicações em que o ambiente operacional impede a implantação de uma estrutura tradicional de rede, como é o caso das operações de defesa, salvamento, sensoreamento de grandes áreas, exploração espacial; aplicações móveis, como é o caso dos sistemas robóticos móveis e distribuídos e aplicações com AGV's (*Automated Guided Vehicles*); e aplicações cuja efemeridade da sua necessidade torna a implantação de uma estrutura de rede economicamente inviável, como é o caso de secretarias de conferências e transmissão remota de informações em eventos

esporádicos.

As redes sem fio devem fornecer as mesmas funcionalidades que as redes cabeadas tradicionais e, adicionalmente, devem proporcionar facilidade de uso, montagem e alteração das configurações, flexibilidade, redução de custos de reconfiguração e mobilidade potencial. Entretanto, as características associadas ao meio físico compartilhado trazem uma série de questões que devem ser consideradas, como é o caso da regulamentação das freqüências de transmissão, da segurança, do controle de acesso, da interferência e da confiabilidade. Em respostas a estas questões, alguns orgãos de padronização criaram grupos de trabalho e propuseram padrões que visam tratar das questões apresentadas. São exemplos de padrões propostos para as redes sem fio: o Bluetooth (BS05), o IrDA (Ass05), o HomeRF (HWG02), o HIPERLAN (Com95) e o IEEE 802.11 (IEE99). Os três primeiros padrões citados são de curto alcance e mais usuais em WPAN's (*Wireless Personal Area Networks*). Os dois últimos são projetados para uma maior área de cobertura e compõem as especificações WLAN's (*Wireless Local Area Networks*). O padrão HIPERLAN (HIgh PErformance Radio LAN) é um padrão europeu definido pelo European Telecommunications Standards Institute (ETSI) no contexto do projeto BRAN (Broadband Radio Access Networks). O IEEE 802.11 é uma padrão para redes sem fio definido pelo grupo de trabalho 11 do comitê de padronização para LAN/MAN do IEEE. O padrão IEEE 802.11 é um padrão americano e que atualmente é o padrão de maior aceitação pelo mercado. Este padrão será apresentado com mais detalhes no capítulo 3.

Uso de Redes sem Fio na Automação

Após o crescimento e popularização do uso das redes sem fio em ambientes de automação de escritório, é possível se observar a ampliação do interesse do uso das redes sem fio como suporte à automação industrial (Net05). O uso de redes sem fio, particularmente as WLANs, traz consigo uma série de novas possibilidades para o ambiente de automação. Dentre estas possibilidades, sistemas robóticos autônomos distribuídos (MR04) e os sistemas de transporte móveis (Tsu86, JB93) têm merecido particular atenção.

Embora de crescente interesse, o uso de redes sem fio na automação ainda fornece uma série de questões que precisam ser tratadas, para que o uso desta classe de rede seja efetivo no meio industrial. A maioria destes desafios estão associados a características não-funcionais que devem ser fornecidas pelas redes em questão, tais como: requisitos de tempo-real, confiabilidade, disponibilidade e segurança (Net05).

Requisitos de tempo-real estão relacionados à previsibilidade no tempo decorrido entre o envio e a entrega da mensagem e à previsibilidade no tempo de acesso ao meio compartilhado. Como este tempo está relacionado à entrega de mensagens fim a fim, os requisitos de tempo-

real devem, adicionalmente, considerar a previsibilidade do tempo de associação e reassociação para estações que estejam se deslocando entre áreas cobertas por pontos de acesso distintos.

Em relação à confiabilidade, deve-se considerar que redes sem fio estão muito mais sujeitas a falhas no envio de mensagens e a perdas de acesso do que as redes cabeadas tradicionais (SVM00). Diversas aplicações não toleram a perda de mensagens, nestes casos protocolos que busquem garantir a retransmissão de mensagens não entregues devem ser utilizados (redundância temporal). Entretanto, a retransmissão de mensagens e o atraso na entrega de informações, podem conflitar com os requisitos de tempo-real existentes para as aplicações. Assim, soluções que busquem equilibrar as necessidades temporais com as necessidades de confiabilidade devem ser investigadas.

A disponibilidade está relacionada ao tempo médio entre a ocorrência de falhas e o tempo necessário para recuperação da falha ocorrida em uma WLAN. Para endereçar estes requisitos, técnicas de tolerância a falhas baseadas em redundância espacial, técnicas de redução de consumo de energia e técnicas de reconfiguração devem ser utilizadas.

Com respeito à segurança, o meio físico compartilhado e sem barreiras físicas torna as redes sem fio potencialmente mais vulneráveis à intrusão. Para endereçar tais requisitos, o uso de protocolos de controle de acesso e encriptação devem ser considerados. Entretanto, de modo similar aos requisitos de confiabilidade, o tratamento da segurança deve ser equilibrado com os requisitos temporais das redes, pois a utilização de complexos algoritmos de encriptação, embora favoreça a segurança, pode comprometer os prazos de entrega de mensagens entre as estações componentes.

2.2 Sistemas de Tempo Real

Uma taxonomia possível para os sistemas computacionais é dividi-los em sistemas transformacionais e sistemas interativos (FFO00). Sistemas transformacionais são aqueles que calculam valores de saídas a partir de valores de entrada fornecidos e, a seguir, finalizam. Este é o caso dos compiladores e programas de cálculo numérico. Por sua vez, um sistema computacional é dito interativo, quando interage permanentemente com o seu ambiente. Dentre os sistemas interativos, destacam-se os chamados sistemas reativos que reagem enviando respostas aos estímulos de entradas que provenham do ambiente no qual se encontram inseridos. Nas aplicações mecatrônicas, os sistemas computacionais utilizados são tipicamente reativos. Na maioria destas aplicações, sensores colhem dados do ambiente, um sistema computacional de controle processa estes dados e mecanismos de atuação agem de forma a modificar o ambiente de acordo

com os resultados do processamento realizado. Quando, além de reagir a estímulos oriundos do ambiente, o sistema computacional tenha de fazê-lo em prazos específicos, está-se diante de uma subclasse dos sistemas reativos, chamados de sistemas computacionais de tempo-real.

Um sistema computacional de tempo-real é um sistema que deve realizar suas funções dentro de limites temporais previamente especificados (prazos ou *deadlines*) (KK97). Nestes casos, o sistema, para ter um comportamento correto, deve não apenas fornecer resultados íntegros (correção funcional ou *correctness*), como deve fazê-lo no momento correto (correção temporal ou *timeliness*).

Sistemas computacionais de tempo-real são de especial relevância para a mecatrônica, na medida em que a sua utilização é extremamente usual em aplicações de controle industrial, automação e robótica, onde, tipicamente, os agentes precisam reagir a mudanças no ambiente, dentro de prazos temporais previamente definidos. Nestas aplicações, a atuação deve se dar em um instante tal que os dados colhidos pelo mecanismo de sensoreamento ainda representem o ambiente mesmo que de forma aproximada, sob pena da atuação não mais ser útil ou até mesmo representar uma ameaça, como é o caso, por exemplo, da atuação realizada por um sistema de controle de caldeira, através de um mecanismo de maçaricos, baseado numa leitura de temperatura desatualizada e que não mais condiza com os valores verificados na prática.

Do ponto de vista dos requisitos de segurança, sistemas de tempo-real podem ser classificados em sistemas de tempo-real brandos (*Soft Real-Time Systems*) e sistemas de tempo-real críticos (*Hard Real-Time Systems*)(FFO00). Um sistema de tempo-real é dito não crítico ou brando se os prejuízos decorrentes de uma falha do sistema é da mesma ordem de grandeza daquele obtido pelo seu funcionamento normal. As aplicações multimídia e os sistemas de comutação telefônica são exemplos típicos de sistemas de tempo-real brandos. Um sistema de tempo-real é dito crítico ou duro quando as conseqüências decorrentes de uma falha excedem em muito os benefícios normais do sistema. Nestes sistemas, falhas podem implicar em perda de dinheiro, tempo ou, até mesmo, vidas humanas. Os sistemas de tempo-real críticos são extremamente comuns na mecatrônica e podem ser identificados em aplicações de controle de veículos aéreos e espaciais, em aplicações de controle de tráfego ferroviário e aéreo, nos sistemas de controle de plantas nucleares e industriais, nas aplicações de controle de unidades de terapia intensiva, nos sistemas de defesa e aplicações militares, dentre outras. Nestes sistemas, os requisitos de confiabilidade são extremamente altos, tipicamente especificados como de 10^{-7} (ou menos) falhas por hora (KSW96).

Os requisitos temporais, por introduzirem mais uma dimensão de complexidade, tornam os sistemas de tempo-real mais propensos a falhas de projeto e construção do que quando comparados aos sistemas concorrentes, móveis e seqüenciais, que não possuam prazos restritivos

para o cumprimento de tarefas. Quando aplicados ao controle e automação, sistemas de tempo-real são ainda mais complexos e propensos a erros sutis, em função do conjunto de interações complexas que tais sistemas realizam com o ambiente no qual atuam.

Adicionalmente, de forma similar ao que acontece com os sistemas computacionais que não possuem requisitos temporais, os projetistas de sistemas de tempo-real vêm adotando o uso de sistemas distribuídos de forma a obter as vantagens inerentes à distribuição (CDK94) no projeto de sistemas de tempo-real (Sta96). Dentre estas vantagens pode-se destacar a inserção de mecanismos de tolerância a falhas para aumentar a confiabilidade e a disponibilidade, que são características fundamentais para sistemas computacionais que possuem altos requisitos de confiabilidade.

Sistemas de Tempo Real sobre Redes sem Fio

Inicialmente, a necessidade de se utilizar canais de comunicação sem fio como suporte a aplicações de tempo-real se deu devido à necessidade de se disponibilizar informações multimídia em dispositivos sem fio. A construção de protocolos que forneçam qualidade de serviço sobre canais sem fio, em particular, em redes aderentes ao padrão IEEE 802.11 vem sendo largamente estudada (CJL02, Zha03, SGZ01). Ainda neste sentido, uma suplementação da especificação, denominada IEEE 802.11E, foi proposta pelo grupo de trabalho E do projeto IEEE 802.11. O IEEE 802.11E fornece a possibilidade de se criar categorias de tráfego para as informações transmitidas. Isto permite que determinadas mensagens sejam identificadas como prioritárias em relação a outras. Assim, aumenta-se a probabilidade de que quadros mais prioritários possam ser transmitidos em detrimento do transporte de quadros de menor prioridade. Por exemplo, às mensagens de correio eletrônico deveriam ser atribuídas classes de baixa prioridade, enquanto que, às mensagens de voz sobre IP poderiam ser atribuídas classes de prioridade mais altas. A IEEE 802.11E é baseada nas redes IEEE 802.11 e fundamenta-se em garantias probabilísticas que são, em diversos casos, insuficientes para as necessidades de previsibilidade demandadas pelos sistemas de tempo-real.

Mais recentemente, as redes sem fio passaram a ser consideradas como suporte a comunicação de sistemas mecatrônicos de tempo real. Redes sem fio foram visualizadas como forma de possibilitar a comunicação entre sistemas de navegação embarcados e autônomos que necessitem trocar informações. Este é o caso, por exemplo, das aplicações de coordenação carro a carro (*car by car*), em sistemas autônomos de controle de trânsito (NS03, NFA$^+$03).

Alguns sistemas, em função de sua imersão em um ambiente dinâmico ou que não seja totalmente previsível, necessitam de flexibilidade o suficiente para possibilitar a sua auto-reconfiguração. Este é o caso, por exemplo, dos sistemas multi-agentes de prospecção de

vulcões. Nestes casos, o sistema deve ser dotado da mobilidade e autonomia características dos sistemas autônomos sem-fio, necessitando do suporte de comunicação das WLANs. Nos casos em que a troca de informações entre os agentes autônomos esteja sujeitas a prazos, cujos cumprimentos sejam essenciais à correta cooperação entre os agentes, a provisão de previsibilidade do tempo de entrega de mensagens se faz necessária (MN99).

2.2.1 Previsibilidade e Confiabilidade em Sistemas de Tempo-Real

Para se resolver o problema do cumprimento de prazos nos sistemas de tempo-real, é insuficiente o aumento da velocidade computacional. O objetivo da chamada computação rápida (*fast computing*) é diminuir o tempo médio de resposta de um dado conjunto de tarefas. Deve se ter em mente, entretanto, que tempos de resposta curtos não dão nenhuma garantia de que todos os requisitos temporais de cada tarefa de um sistema serão cumpridos. Desta forma, mais importante que a rapidez (que é uma grandeza relativa), o ponto fundamental para o projeto de sistemas de tempo-real é a previsibilidade (Sta88). Um sistema de tempo-real é dito previsível no domínio lógico e no domínio temporal quando, independentemente de variações ocorrendo em nível de *hardware*, da carga e de falhas, o comportamento do sistema pode ser antecipado antes da sua execução (FFO00). Assim, a confiabilidade de um sistema computacional de tempo-real está intimamente relacionada à sua previsibilidade, pois quanto mais conhecimento prévio se tem sobre o comportamento do sistema, mais se pode fornecer garantias sobre o seu comportamento.

O uso de testes e simulações, embora útil e necessário para se validar o comportamento do sistema é insuficiente para garantir que o comportamento do sistema construído esteja correto em relação à sua especificação. Isto por que os testes e as simulações consideram um número finito de situações computacionais (entradas e fluxos). Quando se é necessário obter garantias mais fortes das propriedades do sistema, pode-se fazer uso de métodos formais.

2.3 Métodos Formais

Métodos formais são o conjunto de ferramentas, técnicas e linguagens baseados em modelos matemáticos para especificação e verificação de sistemas computacionais (CW96).

O uso de métodos formais já foi considerado no passado como uma alternativa pouco viável para o projeto de sistemas computacionais (CW96). As soluções eram ineficientes e pouco expansíveis, as notações utilizadas eram muito obscuras e as ferramentas eram inadequadas

e difíceis de usar. Existia apenas um número pequeno e pouco representativo de estudos de casos, que não eram suficientes para demonstrar a aplicabilidade dos métodos aos engenheiros de *software* e *hardware*. Além disto, era muito pequeno o número de pessoas com capacitação suficiente para a utilização das ferramentas fora do meio acadêmico.

Mais recentemente, começou-se a observar um quadro mais promissor para o futuro dos métodos formais. As pesquisas vêm crescendo na área e a introdução de ferramentas de verificação automáticas têm possibilitado a adoção destas ferramentas no meio industrial.

A partir da segunda metade da década de noventa, as pesquisas para o uso de métodos formais foram estendidas para a especificação e verificação de sistemas de tempo-real e, atualmente, esta constitui uma área muito ativa de pesquisa, com a definição de diversas técnicas e linguagens, e a construção de várias ferramentas de especificação e verificação.

2.3.1 Especificação de Sistemas de Tempo-Real

Especificar é o processo de descrever o sistema e o conjunto de propriedades que constituem os seus requisitos. Em métodos formais, a linguagem adotada na especificação é baseada em modelos matemáticos, formal e portanto não-ambígua, com sintaxe e semântica claramente definidas.

Durante a especificação, o projetista descreve o sistema de forma rigorosa. Neste processo, o projetista ganha um conhecimento mais completo sobre o sistema que está sendo construído. Desta forma, os principais benefícios alcançados pela utilização de uma linguagem formal de especificação estão relacionados ao fato de que, durante o processo de especificação, os projetistas são contrapostos às incompletudes, inconsistências, ambigüidades e erros existentes no projeto, permitindo a sua correção prematura.

Além disto, uma linguagem formal evita que erros sejam inseridos durante o desenvolvimento do sistema, em função da existência de ambigüidades na sua descrição, decorrentes do baixo rigor formal da linguagem de especificação utilizada.

Linguagens de especificação de sistemas de tempo-real devem permitir a modelagem do comportamento temporal e das restrições temporais requeridas pela aplicação. A abordagem mais largamente adotada para a definição de linguagens de especificação de sistemas de tempo-real tem sido a de realizar extensões sobre linguagens para especificação de sistemas concorrentes e seqüenciais. Nesta abordagem, vale a pena destacar: a linguagem CSP temporizado (*Timed CSP*) (Sch91) baseada no CSP (Hoa85); as estruturas kripke temporizadas (ACD90) baseada nas estruturas Kripke (CGP00); a linguagem ModeChart (JM86) baseada em StateChart (Har87); as redes de petri temporizadas (*Timed Petri Net*) (Wan98) derivadas das redes de petri

(*Petri Net*) (Pet62); e os autômatos temporizados (LPY95).

Em todos estes formalismos, a noção de tempo passa a ser associada aos estados do sistema, assim como às suas transições.

2.3.2 Verificação de Sistemas de Tempo-Real

O uso das técnicas de especificação formal, embora auxiliem no entendimento de um sistema, não garantem que a especificação apresentada satisfaz as propriedades do mesmo. Para isto, é necessária a utilização de alguma técnica de verificação. Verificação é o processo segundo o qual o projetista certifica-se de que as propriedades desejadas são atendidas pela especificação do sistema. Formalmente o processo de verificação consiste em se determinar se $S \models p$ (S satisfaz p), onde S é o modelo de um sistema escrito em alguma linguagem de especificação de sistemas e p é um conjunto de propriedades escrito em uma linguagem de especificação de propriedades.

Desde o início do desenvolvimento das técnicas de verificação formal, duas linhas independentes foram adotadas, baseadas nos artigos pioneiros de (Flo67) e (Hoa69), para o processo de verificação formal de sistemas. O primeiro artigo propunha a validação, através de uma abordagem que se tornou conhecida como prova automática de teoremas. O segundo artigo propunha uma abordagem que se tornou conhecida por verificação de modelos. Ainda hoje, são estas duas abordagens que continuam a ser utilizadas na especificação formal de sistemas computacionais em geral, e em particular, em sistemas computacionais de tempo-real. Nas subseções a seguir são apresentadas estas duas abordagens para sistemas de tempo-real.

2.3.2.1 Prova de Teoremas em Sistemas de Tempo-Real

A primeira abordagem utilizada na verificação formal de sistemas é conhecida por prova de teoremas. Nesta abordagem, o projetista descreve o sistema e as propriedades desejadas como fórmulas de alguma lógica. O processo de verificação consiste em se encontrar uma prova para cada uma das propriedades desejáveis, ou seja, determinar se as propriedades do sistema são teoremas do sistema lógico descrito. Durante a prova dos teoremas podem ser encontradas propriedades inicialmente não vislumbradas e lemas intermediários. Embora a prova de teoremas possa ser feita manualmente, como em (PSvH99), diversos têm sido os trabalhos desenvolvidos que buscam permitir a sua prova automática.

A prova automática de teoremas é aquela que é feita através do uso de ferramentas automatizadas. Estas ferramentas podem ser totalmente automáticas, quando buscam realizar a prova sem qualquer intervenção humana, ou podem ser interativas, quando os projetistas intervêm,

definindo os caminhos de prova a serem seguidos.

Alguns problemas derivados do uso desta abordagem são: os problemas de complexidade dos algoritmos e da explosão combinatória do espaço de busca, principalmente nas provas completamente automáticas; e a lentidão, principalmente quando a prova é interativa. Um outro problema característico da prova automática de teoremas é que a linguagem utilizada difere muito das linguagens normalmente utilizadas para a descrição e implementação do *software*. Isto pode resultar numa quebra de modelo, que se mal gerenciada, pode levar a prova e verificação de um sistema e a descrição e implementação de outro, que embora semelhantes não sejam equivalentes. Além disto, estas linguagens, em geral, demandam mão de obra altamente especializada, tornando custosa a sua aplicação.

Uma característica positiva dos provadores de teoremas é a sua habilidade em tratar sistemas de transição de espaços infinitos.

O uso de provadores automáticos de teoremas para verificação de sistemas de tempo-real tem sido possível através da inclusão de predicados que modelam limites temporais. Esta é a abordagem utilizada em (Sha93), que usa o PVS, uma ferramenta de verificação de sistemas disponível em (PVS03), para verificar um algoritmo de exclusão mútua e um controlador de cruzamento de ferrovias.

2.3.2.2 Verificação de Modelos em Sistemas de Tempo-Real

Verificação de modelos é uma técnica automatizada de verificação de sistemas. Em função disto, esta é uma técnica de especial interesse na indústria. Nesta abordagem, propriedades são expressas em uma lógica proposicional temporal e os sistemas são modelados como sistemas de transição de estados finitos. Um algoritmo de busca eficiente é implementado de forma a determinar se a especificação é verdadeira no sistema de transição em questão. Em outras palavras, um sistema de transição é verificado de forma a se definir se ele é um modelo para as propriedades.

Os verificadores de modelo podem ser utilizados para verificar especificações parciais, e, portanto, fornecer informações úteis sobre a correção de um sistema que ainda não esteja completamente especificado. Uma outra vantagem dos verificadores de modelo é que tais ferramentas fornecem contra-exemplos, quando uma determinada propriedade não é satisfeita, auxiliando o desenvolvedor na correção de erros. A principal desvantagem associada aos verificadores de modelos é a inabilidade destas ferramentas em tratar sistemas de transição de estados infinitos.

As técnicas de verificação de modelos têm experimentado fortes evoluções. A principal

delas foi, possivelmente, o desenvolvimento de algoritmos simbólicos de verificação de modelos (McM92). Esta técnica permitiu que os sistemas, que eram inicialmente modelados como um conjunto de estados associados a uma lista de adjacências entre estes estados, passassem a ser representados através de diagramas de decisão binárias ordenados (OBDDs) (Bry86). Isto permitiu que sistemas com mais de 10^{20} estados fossem verificados automaticamente, em oposição ao limite de 10^5 estados dos primeiros verificadores. Outras técnicas de otimização foram também desenvolvidas como, por exemplo, a construção dinâmica (*on the fly*) e guiada da representação de estados do sistema (GLMR05).

Devido a sua potencial aplicabilidade na indústria, verificação de modelos para sistemas de tempo-real é uma técnica particularmente ativa em métodos formais, e variações de algoritmos de verificação, de linguagens de especificação de sistemas e de linguagens de especificação de propriedades podem ser encontrados em diversos trabalhos (ACD90, CC95, LPY95, FGK96, YSSC93, TY99, LS01, RK02).

2.4 Métodos Formais e Redes sem Fio

Apesar do domínio abrangente da especificação IEEE 802.11 e da existência de várias ferramentas de verificação de modelos, não existem muitos trabalhos que tratem da formalização de sua especificação e da verificação formal de suas propriedades. Pode-se encontrar na própria especificação do padrão (IEE99) uma descrição formal em SDL (*Specification and Description Language*) (IT00). Esta especificação pode ser usada por simuladores e ajudam na tarefa de identificação de situações de erros (Dol03). Embora existam alguns estudos para verificação de modelos de especificações em SDL (BDHS00), a utilização desta especificação do protocolo para verificação, não seria possível devido ao seu alto grau de detalhamento o que implicaria na explosão de espaços de estados. A especificação apresentada tem grande foco no comportamento interno das tarefas do protocolo, ao invés de focar na interação entre as estações.

Outros cenários de simulação têm sido usados para o IEEE 802.11(Cro97). No entanto, a simulação, apesar de melhorar o entendimento do protocolo, não pode fornecer garantias de sua correção, visto que o comportamento do protocolo é simulado para um número finito e não exaustivo de situações, mesmo quando se considera apenas um número de cenários limitados e previamente definidos.

Um trabalho que apresenta uma especificação formal para o padrão IEEE 802.11 com o intuito de provar sua correção foi publicado recentemente (YVM02). Esta especificação, contudo, possui algumas limitações. Em primeiro lugar, o protocolo não foi considerado de maneira in-

tegrada. O padrão IEEE 802.11 estabelece duas funções de coordenação: centralizada (PCF — *Point Coordination Function*), de implementação opcional, e distribuída (DCF — *Distributed Coordination Function*), de implementação obrigatória. Estas funções foram tratadas apenas isoladamente, quando na prática elas devem ocorrer de forma alternada no tempo. Em segundo lugar, a formalização se concentrou apenas na especificação. A verificação foi feita de maneira informal e não-automatizada. Por fim, como não há o conceito de tempo na abordagem utilizada, a prova de correção concentrou-se apenas em propriedades de segurança (*safety*) e progresso (*liveness*). Assim, não foram consideradas as propriedades temporais (*timeliness*). Propriedades estas, fundamentais quando se deseja projetar sistemas de tempo-real.

Um trabalho ainda mais recente que trata da verificação formal da camada MAC do IEEE 802.11 é descrito em (KNS02). Neste artigo os autores combinam a verificação do protocolo através de uma combinação de verificação não-probabilística com uma verificação probabilística utilizando uma ferramenta chamada PRISM (Probabilistic Symbolic Model Checker) (KNP02). Entretanto, verificações probabilísticas são, muitas vezes, insuficientes para as aplicações de tempo-real, visto que em sistemas críticos a mera probabilidade de cumprimento de um prazo de entrega de mensagem é uma hipótese fraca, pois o sistema necessita de determinismo no cumprimento dos limites temporais especificados. Adicionalmente, o artigo trata apenas do comportamento do protocolo durante a utilização da DCF, focando em verificar qual a probabilidade de ocorrência de *livelocks* em determinadas condições e configurações de rede.

CAPÍTULO 3

A Especificação 802.11

É espantosamente óbvio que nossa tecnologia excede nossa humanidade.

—ALBERT EINSTEIN (Físico e Humanista Alemão)

O termo IEEE 802.11 diz respeito a uma família de especificações desenvolvida pelo grupo de trabalho 11 do Comitê de Padronização de LANs/MANs do IEEE com o propósito de obter conectividade sem fio para estações fixas, portáteis e móveis em uma rede local. O desenvolvimento deste padrão foi iniciado em 1990 e a sua versão atual foi disponibilizada em 1999 (IEE99).

A figura 3.1 apresenta a relação entre os diversos padrões da familia IEEE802.x, com destaque para a posição do padrão IEEE 802.11.

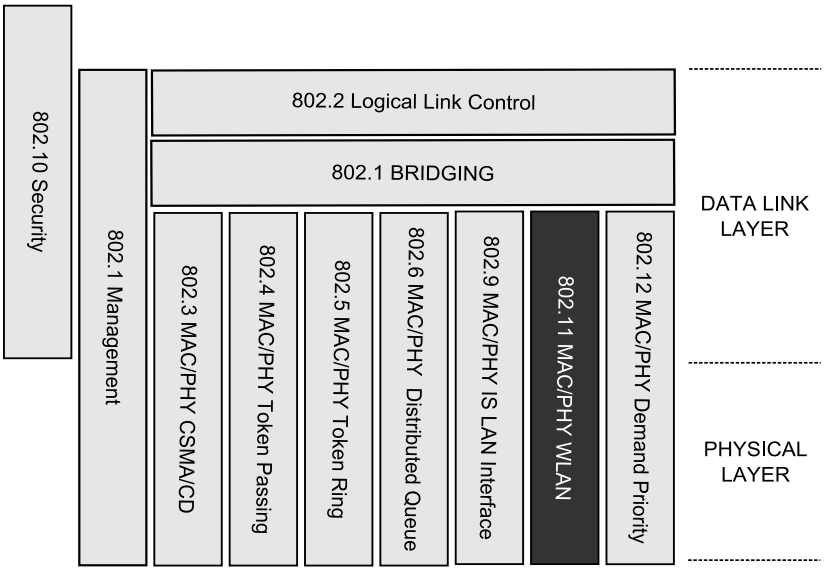

Figura 3.1 Família IEEE 802.x

Como todos os padrões da família IEEE 802.x, o padrão .11 especifica as operações da camada física (PHY) e da camada de acesso ao meio (MAC). Pelo fato da especificação descrever

as funções típicas da camada *Ethernet* em WLAN's, o IEEE 802.11 é popularmente chamado de *Ethernet* sem fio.

Em linhas gerais, a subcamada MAC deste protocolo é responsável por uma série de serviços, tais como autenticação, segurança, fragmentação e remontagem. Dentre os serviços especificados na subcamada MAC do protocolo pode-se destacar o serviço de controle de acesso ao meio. Este serviço é composto por um conjunto de regras que definem como as estações que compõem uma rede padrão IEEE 802.11 acessam o meio e qual são os quadros de controle trocados durante a entrega de mensagens. O conjunto de regras que coordena o acesso ao meio físico denomina-se função de coordenação (CF — *Coordination Function*).

Por sua vez, a subcamada PHY é responsável por tratar dos aspectos de conversão dos dados fragmentados pela camada MAC em sinais a serem trasmitidos e da transmissão e recepção efetiva destes sinais.

Este capítulo apresenta a arquitetura e o funcionamento das funções de coordenação da especificação. A seção 3.1 apresenta as características da arquitetura das redes sem fio IEEE 802.11. A seção 3.2 apresenta o formato dos quadros definidos para o padrão. Na seção 3.3 descreve-se em linhas gerais a subcamada PHY do protocolo. Finalmente, a seção 3.4 é dedicada à apresentação da subcamada MAC do IEEE 802.11.

3.1 Arquitetura do IEEE 802.11

Do ponto de vista da sua arquitetura, o IEEE 802.11 é composto por um conjunto de componentes que buscam fornecer às camadas superiores da rede transparência no que diz respeito à mobilidade de suas estações.

O bloco básico de construção de uma rede IEEE 802.11 é o *basic service set* (BSS). Um BSS é definido como um conjunto de estações controladas por uma única função de coordenação do uso do meio. A área geográfica dentro da qual as estações componentes de um BSS conseguem se comunicar com as outras é denominada *basic service area* (BSA). A dimensão de uma BSA é dependente das características de propagação do sinal no meio. Caso uma estação mova-se para fora da área de sua BSA, ela não mais poderá se comunicar com os membros de seu BSS. Em tese, quando uma estação está em uma BSA ela pode se comunicar diretamente com qualquer uma das estações que compõem o BSS. Entretanto, devido aos aspectos geográficos, à interferência de outros BSSs próximos que usem as mesmas características da camada física ou à degradação do meio físico, algumas estações podem parecer estar ocultas em relação a outras. Este problema é conhecido por problema do nó oculto e caracteriza-se

pela existência de uma estação alcançável pelo sinal de um participante do BSS e inalcançável pelo sinal de outra estação participante do mesmo BSS.

Um BSS pode apresentar três tipos de topologias:

- redes independentes;
- redes com infra-estrutura;
- redes estendidas.

Um BSS independente (IBSS) é a topologia mais simples das redes 802.11. Um IBSS é formado por um conjunto de estações 802.11 que se comunicam diretamente umas com as outras, sem apoio de qualquer infra-estrutura de rede. O menor IBSS possível é aquele formado por apenas duas estações sem fio que se comunicam. As estações participantes de um IBSS devem estar em uma área que permita a comunicação direta entre elas, visto que toda a comunicação é realizada ponto a ponto. Tipicamente os IBSSs são compostos por um número pequeno de estações e destinam-se a realizar uma tarefa específica, possuindo curto tempo de vida. Devido às características apresentadas, os BSSs independentes são popularmente conhecidos por redes sem fio *ad-hoc*. Na figura 3.2 pode-se visualizar a arquitetura típica de um IBSS e de sua correspondente BSA (a área da BSA é representada pela linha ao redor das estações).

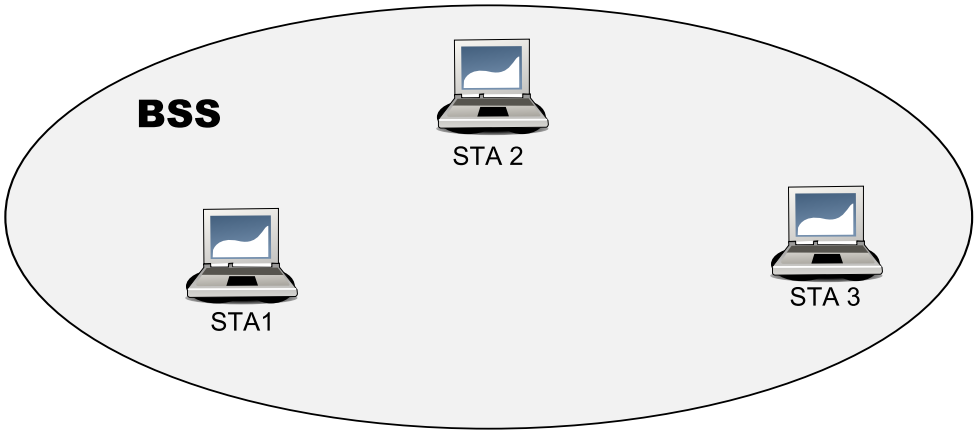

Figura 3.2 Arquitetura de um BSS Independente

Um BSS com infra-estrutura é uma topologia que necessita de uma estação especializada conhecida por AP (*Access Point*). O AP é responsável por todas as comunicações entre as estações participantes do BSS. Deste modo, toda comunicação realizada entre duas estações de um BSS com infra-estrutura é realizada em dois saltos: a estação cliente envia a mensagem

para o AP, e o AP reenvia esta mensagem para a estação destino. Nesta topologia, a BSA corresponde a todos os pontos no espaço onde os sinais do AP podem ser recebidos. Na figura 3.3, é possível se verificar a configuração de uma rede com infra-estrutura, com destaque para a estação que atua como AP.

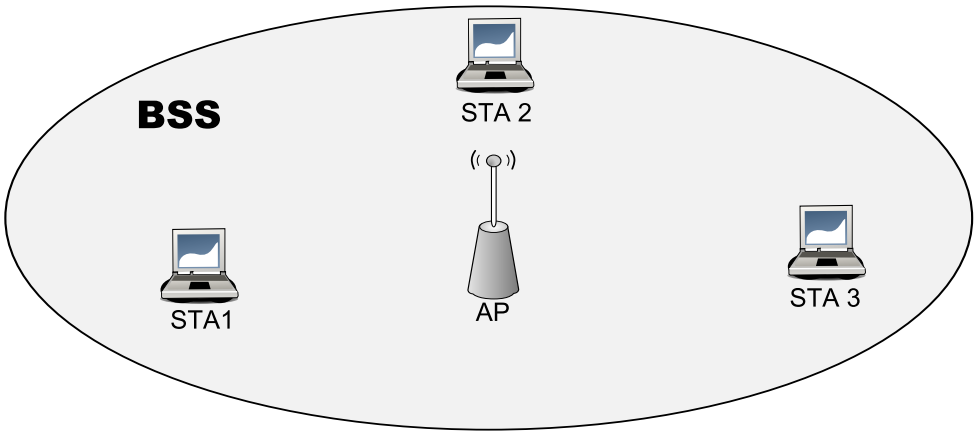

Figura 3.3 Arquitetura de uma Rede com Infra-estrutura

Mesmo com o uso de um AP, as redes IEEE 802.11 estão sujeitas a limitações de tamanho determinadas pelo alcance dos sinais utilizados na camada física. Como forma de superar tais limitações, pode-se unir um conjunto de BSSs numa topologia denominada *extended service set* (ESS) ou redes estendidas. Um ESS é criado através da união de diversos BSSs por um componente arquitetural denominado sistema de distribuição (DS, do inglês *distribution system*). O DS pode ser visto como um *backbone* que interliga os APs e fornece um determinado conjunto de serviços, tais como o de transporte dos pacotes da camada MAC (MSDU — MAC *service data unit*) entre os diferentes BSSs que compõem a ESS. O IEEE 802.11 não define qual a tecnologia que será usada no sistema de distribuição, o que permite, por exemplo, que o DS seja implementado tanto sobre um meio sem fio, como através de cabos por redes *ethernet token bus*, *token ring*, FDDI, entre outras. A figura 3.4 ilustra a composição de uma ESS.

Finalmente, para a conexão de uma rede sem fio IEEE 802.11 com redes tradicionais com fio, um último componente arquitetural é necessário. Este componente é conhecido por portal. O portal é a entidade lógica que integra um DS com redes não IEEE 802.11 (Cro97). O portal atua como uma *bridge*, do ponto de vista lógico. É possível que um mesmo dispositivo físico acumule as funções de AP e de portal. Esta arquitetura é ilustrada na figura 3.5.

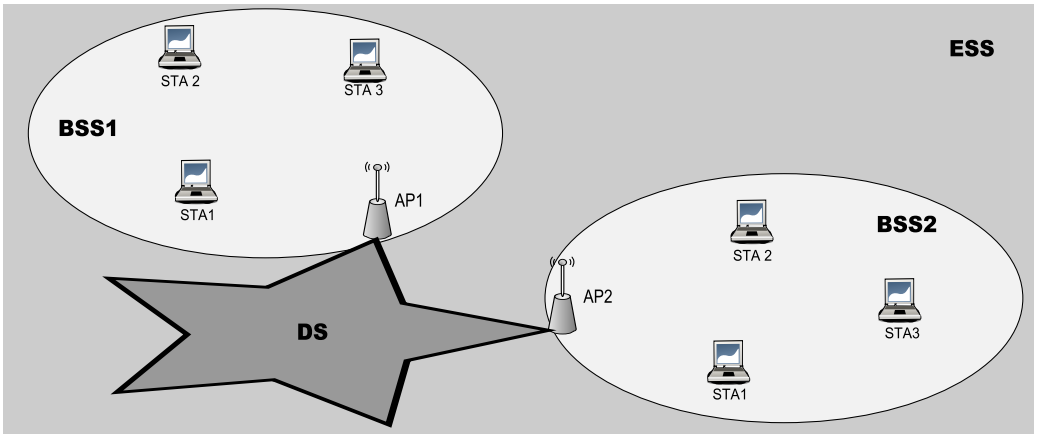

Figura 3.4 Arquitetura de uma ESS(*Extended Service Set*)

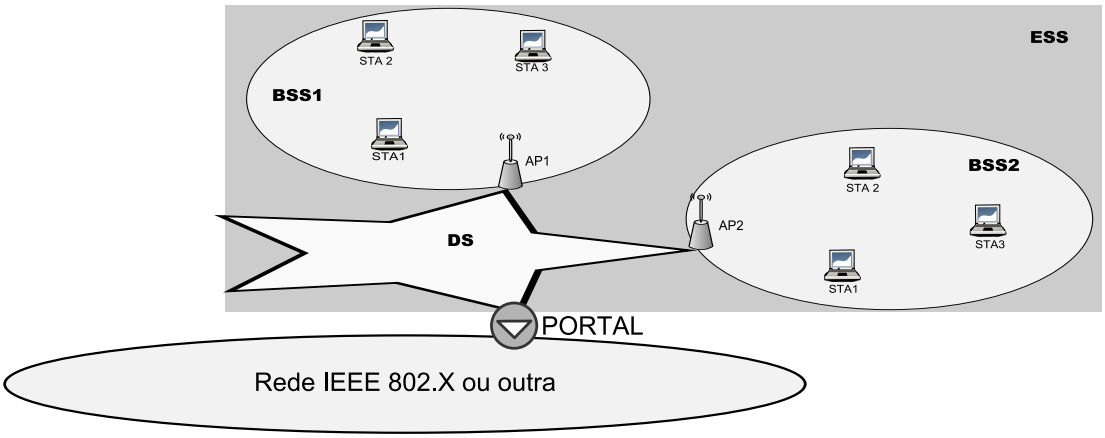

Figura 3.5 Conexão de Redes IEEE 802.11 a redes cabeadas através de um portal

3.2 Quadros do Protocolo IEEE 802.11

A especificação IEEE 802.11 define o formato válido dos quadros que devem ser utilizados para o controle e troca de mensagens entre as estações participantes de uma das topologias de rede apresentadas na seção 3.1. A figura 3.6 exibe o formato genérico dos MPDUs (quadros) do IEEE 802.11.

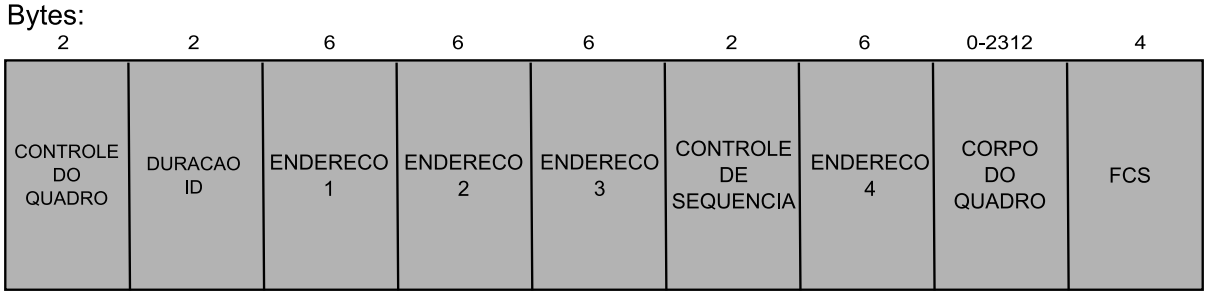

Figura 3.6 Quadro de dados genérico do IEEE 802.11

Os MPDUs são formados pelos seguintes componentes: cabeçalho (MAC *Header*), corpo e código de verificação de integridade (FCS — *Frame Check Sequence*).

No cabeçalho, o protocolo armazena informações de controle (*Frame Control*), informações de controle de seqüência para remontagem (*Sequence Control*), campos de endereçamento (*Address1* a *Address4*), dados de identificação do quadro e de duração para a reserva de uso do meio (*Duration/ID*). O campo de controle é composto por dois octetos. Destes dois octetos, os dois primeiros bits (B0 e B1) são utilizados para transportar informações de versão do protocolo. Os bits seguintes (B2 e B3) indicam o tipo do quadro (controle, dados ou gerenciamento) e os quatro próximos (B4 a B7) indicam o sub-tipo específico do quadro em questão. Os valores assumidos pelos campos de tipo e sub-tipo dos quadros são apresentados na tabela 3.1. Os oito bits seguintes tranportam informações de direcionamento do quadro (*To DS Field* e *From DS Field*), de controle de fragmentação (*More Fragments Field*), de retransmissão (*Retry Field*), de gerenciamento de energia (*Power Management Field* e *More Data*), de controle de criptografia (*WEP Field*) e de controle de ordenação em quadros fragmentados (*Order Field*).

Após o campo de controle, o cabeçalho possui o campo *Duration/ID*. Este campo possui dois octetos e nos quadros de controle PS-Poll transporta informações sobre a estação que transmitiu o quadro. Para os demais quadros, o campo transporta informações de reserva de uso do meio utilizadas pelo NAV. Nos quadros transmitidos durante o CFP, este campo é ajustado no valor 32.768. Em seguida o cabeçalho armazena três campos de endereçamento de 6 octetos cada um. Estes campos, em conjunto com o campo *Address4*, são usados para identificar a

fonte e o destino de um quadro, além de transportar a identificação da BSS destino dentro da ESS e os endereços das estações intermediárias. Os próximos dois octetos (campo *Sequence Control*) contêm a numeração dos fragmentos de um MSDU e são usados na remontagem e na detecção de quadros duplicados. Seguindo o campo *Sequence Control*, encontra-se os 6 octetos do campo *Address4*.

Após o cabeçalho vem o corpo do quadro, um componente de tamanho variável e conteúdo dependente do tipo da mensagem. O tamanho do corpo do quadro pode variar entre 0 e 2312 octetos. Finalmente, após o corpo, o quadro traz um código de verificação de integridade composto por 4 octetos e formado por um valor gerado através do polinômio padrão do IEEE de 32-bit para CRC (*Cyclic Redundancy Code*). O CRC é calculado sobre todos os octetos do cabeçalho e corpo do quadro.

Para a transmissão de informações e controle de acesso ao meio é utilizado, além dos quadros de dados e de controle, o quadro de gerenciamento *beacon*. Este quadro é utilizado para gerenciamento de sincronismo e alternância das funções de coordenação DCF e PCF a serem explicados nas seções 3.4.1 e 3.4.2.

3.3 A camada física no IEEE 802.11

O IEEE 802.11 apresenta três descrições de padrões a serem utilizados na camada física. Dois deles dizem respeito a especificações para sinais de rádio freqüência (o FHSS e o DSSS) e um deles apresenta a especificação da camada física para infra-vermelho (IR). Atualmente, as técnicas de espalhamento espectral com sinais de rádio são as mais largamentes utilizadas por prover maior segurança, integridade e confiabilidade, empregando para isto um maior consumo de banda.

O FHSS (*Frequency Hopping Spread Spectrum*) prevê a transferência de informação a uma taxa de 1 Mbit/s (opcionalmente os fornecedores podem fornecer suporte operacional a taxa de 2 Mbit/s). Em linhas gerais, o FHSS divide a banda passante em micro bandas. O transmissor e o receptor utilizam uma desta bandas por um intervalo de tempo e em seguida 'saltam' (*hop*) para uma nova micro banda. O objetivo deste comportamento é permitir a coexistência de diversas redes dentro de uma mesma área, através da adoção por cada uma destas redes de padrões pseudo-aleatórios e distintos de seqüências de variações de utilizações das micro-bandas, conhecidas por seqüência de saltos. A sincronização entre o transmissor e o receptor através de uma mesma seqüência de saltos resulta na manutenção de um canal lógico para estas estações.

O DSSS (*Direct-Sequence Spread Spectrum*) é uma técnica de espalhamento de sinal que

Tabela 3.1 Combinações Válidas de Tipos e Sub-Tipos no IEEE 802.11

B2 - B3	**Tipo**	**B4 - B7**	**Sub-Tipo**
00	Gerenciamento	0000 - 0011	Gerenciamento de associação e reassociação de estações
00	Gerenciamento	0100 - 0101	Gerenciamento de busca de BSS
00	Gerenciamento	0110 - 0111	Reservados
00	Gerenciamento	1000	*Beacon*
00 00	Gerenciamento Gerenciamento	1001 1001	ATIM - *Announcement Traffic Indication Message*. Usado para gerenciamento de energia em IBSSs
00	Gerenciamento	1010	Desassociação
00	Gerenciamento	1011 - 1100	Gerenciamento de autenticação
00	Gerenciamento	1101 - 1111	Reservados
01	Controle	0000 - 1001	Reservados
01	Controle	1010	PS-Poll - Usado para gerenciamento de energia
01	Controle	1011	*Request To Send* (RTS)
01	Controle	1100	*Clear To Send* (CTS)
01	Controle	1101	*Acknowledgment* (ACK)
01	Controle	1110	*Contention-Free End* (CF-End)
01	Controle	1111	CF-End + CF-Ack
10	Dados	0000	Dados
10	Dados	0001	Data + CF-Ack
10	Dados	0010	Data + CF-Poll
10	Dados	0011	Data + CF-Ack + CF-Poll
10	Dados	0100	Null (no data)
10	Dados	0101	CF-Ack (no data)
10	Dados	0110	CF-Poll (no data)
10	Dados	0111	CF-Ack + CF-Poll (no data)
10	Dados	1000 - 1111	Reservado
11	Reservado	0000 - 1111	Reservado

busca permitir a comunicação em taxas superiores a 2 Mbit/s, além da ampliação da BSA. O DSSS fundamenta-se na idéia de dispersão de um sinal original de grande amplitude e baixa largura de banda, em um sinal de baixa amplitude que ocupe uma larga faixa de freqüência para transmissão. Este processo é realizado pelo *spreader*. Posteriormente, durante a recepção, o sinal original é remontado. O processo de remontagem é chamado de correlação. Durante a correlação o sinal espalhado pela banda adquire seu formato original de alta intensidade e baixo espalhamento. Assim, ruídos injetados durante a transmissão, tipicamente de baixa largura de banda e alta intensidade, ao serem correlacionados, são minimizados em intensidade e espalhados pela banda. A modulação do DSSS é feita através da divisão da banda disponível total em diversos canais de transmissão independentes. O sinal a ser transmitido é então codificado pelo uso de uma sequência de bits conhecida por *chip*. O *chip* é um sinal binário utilizado apenas para a codificação. Durante a modulação dos dados o *spreader* executa operações OR (XOR) entre os bits do *chip* e os bits de dados. Cada bit de dado é representado pelo total de bits do *chip*, que serão transmitidos cada um por um canal de transmissão de modo independente. Na recepção, pode-se utilizar técnicas estatísticas para recuperar dados que tenham sido alterados durante a transmissão, sem a necessidade de retransmissão. Deste modo, quanto maior o *chip* maior será a probabilidade de recuperação da informação original. Entretanto, quanto maior o *chip*, maior será a banda requerida. O 802.11 utiliza para o espalhamento a seqüência de Barker de 11 *chips* (1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0). Em seguida, cada um dos bits codificados é transmitido em um dos 11 canais de 11 MHz resultantes da divisão da banda original.

A especificação de infravermelho (IR — *Infra-Red*) utiliza freqüências muito altas no espectro eletromagnético, pouco abaixo da luz visível (comprimento de onda na faixa de 850 a 950 *nm*). De modo similar à luz, o infravermelho não pode se propagar através de objetos opacos. Por isto, esta especificação é útil para ser utilizada em áreas fechadas e em PANs (*Personal Area Networks*). Sistemas infravermelhos diretos de baixo custo somente conseguem cobrir distâncias em torno de 1,5 m, enquanto para alguns receptores mais sensíveis a área de cobertura é, tipicamente, de 20 metros.

Independente da especificação, a camada PHY sempre irá fornecer um mecanismo de sinalização para indicar se o meio encontra-se livre ou ocupado (CCA — *clear channel assessment function*). Este sinal será utilizado pelo mecanismo de sensoreamento do meio para definir o estado atual do meio (*physical carrier-sensing*).

3.4 Controle de Acesso ao meio no IEEE 802.11

Um dos objetivos do padrão 802.11 é prover um mecanismo de acesso ao meio, que seja justo e que leve em consideração a inabilidade das estações móveis em detectar colisões, do modo como as estações de redes com infra-estrutura de cabos fazem. Esta inabilidade está relacionada à característica dos equipamentos eletro-eletrônicos encarregados de prover a efetiva comunicação sem fio. Assim, por exemplo, o protocolo deve considerar a existência de estações equipadas com estações de rádio que só podem escutar as transmissões realizadas por outras estações, caso não esteja ela própria transmitindo (rádio *half-duplex*). Esta é uma hipótese necessária, principalmente devido ao fato de rádios *full-duplex*, capazes de transmitir e escutar transmissões simultaneamente, possuírem um custo muito mais elevado que os rádios *half-duplex*.

Além de considerar as estações como *half-duplex*, o protocolo deve considerar que o mecanismo de controle especificado deve causar o menor *overhead* possível. É visando este objetivo, que o padrão IEEE 802.11 descreve duas funções de coordenação do uso do meio: uma Função de Coordenação Centralizada (PCF — *Point Coordination Function*) e uma Função de Coordenação Distribuída (DCF — *Distributed Coordination Function*).

Na PCF, uma das estações atua como árbitro, e define o momento em que cada uma das estações da rede poderá enviar quadros. Na DCF não há uma estação que atue como árbitro. Ao tentar enviar um quadro, cada estação disputa o meio com as demais. Esta disputa pode levar a colisões.

Quando a rede opera sob a função PCF, diz-se que ela está no Período Livre de Disputa (CFP — *Contention Free Period*), uma vez que a arbitragem do meio impede colisões. Quando a rede opera sob a função DCF, diz-se que ela está no Período de Disputa (CP — *Contention Period*). Em uma rede que suporta a PCF, os dois métodos de acesso se alternam em ciclos, onde um CFP é seguido por um CP. Períodos livres de disputa apenas são suportados em redes com infra-estrutura, cabendo ao AP atuar como responsável pelo escalonamento do uso do meio.

3.4.1 DCF — Função de Coordenação Distribuída

A Função de Coordenação Distribuída (DCF) é o método básico de acesso ao meio da IEEE 802.11 e, portanto, deve ser implementado em todas as estações participantes de uma rede IEEE 802.11. A DCF baseia-se num mecanismo do tipo LBT (*Listen Before Talk*) conhecido por CSMA-CA (*Carrier Sense Multiple Access / Collision Avoidance*). O CSMA/CA é uma

protocolo que se fundamenta no sensoriamento do meio antes do envio de mensagens (*Carrier Sense*) associado a um mecanismo para reduzir a probabilidade da ocorrência de colisões (*Collision Avoidance*).

A DCF não oferece nenhum mecanismo para garantir retardos máximos para estações que forneçam serviços com garantias temporais (política de *melhor esforço*) (Cro97).

Na DCF, uma estação, que deseje enviar um quadro deve aguardar um determinado período de inatividade do meio antes de fazê-lo. Findo este período, caso o meio permaneça livre, a estação irá enviar o quadro e esperar uma confirmação de seu recebimento (ACK). Uma vez iniciada a transmissão de um quadro, a estação continua o envio até ter enviado todo o quadro, mesmo que ocorra colisão. Caso o meio não esteja livre, a estação irá adiar a transmissão do seu quadro até o final da transmissão corrente. Após o final da transmissão a estação irá executar um procedimento de *backoff* e só então a estação tentará enviar o quadro novamente. A estação repete este procedimento até que receba uma confirmação de recebimento de seu quadro. Os detalhes do funcionamento do mecanismo de sensoriamento do meio, do espaçamento entre quadros, do mecanismo de *backoff* e do funcionamento específico dos modos de operação da DCF são descritos a seguir.

3.4.1.1 Função de Sensoriamento do Meio

A função de sensoriamento do meio é utilizada para determinar se o meio físico está disponível para utilização pela estação. Esta função é realizada pela associação de um mecanismo de verificação na subcamada física (*physical carrier-sensing*) com um mecanismo de verificação na subcamada MAC (*virtual carrier-sensing*).

Na subcamada física, o mecanismo de verificação é realizado através da CCA. Esta função se baseia na percepção da existência de sinais físicos no meio, sendo extremamente dependente do tipo do sinal físico utilizado. Normalmente, as estações móveis são incapazes de realizar a escuta do meio, simultaneamente enquanto transmitem, devido às limitações dos circuitos eletrônicos que as compõem. Circuitos eletrônicos que possuem a capacidade de perceber o estado do meio ao mesmo tempo que transmitem são inviáveis do ponto de vista econômico. Outro problema relacionado ao mecanismo de verificação da camada física, é a sua vulnerabilidade ao problema do nó oculto.

Na subcamada MAC, a verificação do estado do meio é realizada através da manutenção de um temporizador local a cada estação, denominado NAV (*Network Allocation Vector*). O NAV armazena a quantidade de tempo que deve decorrer até que o meio esteja novamente livre. As estações atualizam os seus NAVs de acordo com informações de alocação, fornecidas pelas

estações que no momento detém o controle do meio de transmissão. Estas informações são inseridas no cabeçalho de cada um dos quadros trocados durante a comunicação. Uma estação irá reconhecer a existência de um novo valor para o seu NAV quando o campo de duração do período de alocação, presente no quadro que estiver sendo transmitido, for maior que o valor atual do NAV da estação e, além disso, a estação não seja a estação destino do quadro.

Enquanto nenhum novo valor de NAV for fornecido, as estações decrementam os NAVs até que eles cheguem a zero. Quando o valor do NAV é diferente de zero, o mecanismo de *virtual carrier-sensing* considera que o meio está ocupado; quando o valor do NAV alcança o valor 0, o mecanismo considera que o meio está livre.

Para que uma estação considere que o meio está livre, é necessário que tanto o *physical carrier-sensing* quanto o *virtual carrier-sensing* registrem que o meio está livre. Ou seja, o meio é considerado livre pela estação quando o CCA indique ausência de sinal e o valor do NAV da estação seja zero.

3.4.1.2 Espaçamento entre Quadros

Os períodos de espaçamento entre quadros (IFS — *interframe spacing*) são intervalos temporais entre as transmissões de cada quadro, no qual o meio deve permanecer inativo. Uma das principais funções dos IFSs é fornecer mecanismos para acesso prioritário ao meio. Quanto menor o IFS, maior a prioridade do acesso, visto que a estação necessita aguardar menos tempo antes de tentar utilizar o meio. O valor dos IFS é definido de acordo com as características da subcamada física.

Por ordem inversa de tamanho e direta de prioridade, são três os tipos de IFSs definidos para o DCF:

- SIFS (*Short Interframe Spacing*),
- DIFS (*DCF Interframe Spacing*),
- EIFS (*Extended Interframe Spacing*).

O SIFS é o menor período de espaçamento entre quadros. Ele deve ser utilizado quando existem trocas de quadros que estão sendo realizadas em uma sequência atômica. O SIFS é utilizado, por exemplo, antes do envio de um quadro ACK por uma estação que tenha recebido um quadro com sucesso.

O DIFS é o menor tempo de inatividade a ser aguardado por estações que desejam utilizar o meio durante o período de disputa. De modo genérico, uma estação que deseja transmitir sob

a DCF realiza o sensoriamento do meio e, ao verificar que este se encontra inativo, permanece escutando o meio por um período DIFS. Caso o meio permaneça inativo durante todo este tempo, a estação transmite o quadro (MPDU — *MAC Protocol Data Unit*) desejado.

O EIFS é o maior dos períodos de espaçamento entre quadros. Um EIFS é utilizado durante o DCF quando a camada física indicar que a transmissão resultou em erro. Esta detecção é realizada através da verificação do campo FCS (*frame check sequence*), que armazena um valor de CRC (*cyclic redundancy check*) anexado ao pacote. Este erro pode indicar uma interferência no meio e, portanto, a impossibilidade temporária de transmissão, mesmo que o mecanismo de sensoriamento indique que o meio está livre.

As durações dos períodos de espaçamento entre quadros são dependentes das características de transmissão do meio físico, representadas através do valores *aSIFSTime* e *aSlotTime*. Como exemplo, o tempo de duração do SIFS é dado por $SIFS = aSIFSTime$, enquanto o tempo de duração do DIFS é dado por $DIFS = aSIFSTime + 2 \times aSlotTime$.

3.4.1.3 Mecanismo de *Backoff*

O *backoff* é a técnica utilizada pelo DCF para realizar a tarefa de evitar colisão (CA) do mecanismo CSMA/CA. O *backoff* é aplicável no momento em que a ocorrência de colisões seja mais provável de acontecer. Isto ocorre no instante em que o meio torna-se ocioso após ter estado um período ocupado, pois podem existir diversas estações aguardando que o meio fique disponível para iniciar a transmissão.

No mecanismo de *backoff* uma estação que deseje iniciar uma transmissão deve utilizar a função de sensoriamento do meio para verificar se o mesmo se encontra livre. Caso o meio esteja ocupado, a estação deverá aguardar até que o meio se torne ocioso. Em seguida, a estação irá aguardar por um DIFS, caso a transmissão do último quadro tenha acontecido sem erros, ou um EIFS, caso tenha acontecido algum erro reportado pela camada física. Então, a estação selecionará, aleatoriamente, um valor entre 0 e o valor atual da janela de contenção (CW — *Contention Window*). O valor inicial da CW é definido por *aCWMin*. Este valor será utilizado para definir o valor do tempo de *backoff* de acordo com a equação $Backoff Time = Random(0, CW_{atual}) \times aSlotTime$. A estação passa a utilizar a função de sensoriamento do meio para determinar se existe alguma atividade do meio durante cada intervalo de tempo (*aSlotTime*) do *backoff*. Para cada um destes intervalos, caso o meio mantenha-se em inatividade, o tempo de *backoff* é diminuido de um *aSlotTime*. Se em algum momento, durante um intervalo de tempo do *backoff*, o meio voltar a ficar ocupado, o tempo de *backoff* é congelado, isto é, não é diminuido de valor algum. Neste caso, a estação volta a aguardar que o

meio se torne ocioso e novamente espera pela passagem de um DIFS ou EIFS, conforme o caso, continuando com o processo de decremento do tempo de *backoff*. Este procedimento é mantido até que o tempo de *backoff* chegue a zero. Neste instante, a estação inicia a transmissão.

Caso exista a ocorrência de colisão, detectada pela ausência do quadro de confirmação, a estação reinicia o procedimento de *backoff*, alterando o valor da CW para o próximo valor de potência inteira de 2 menos 1 ($2^n - 1$), ou até o limite máximo de *aCWMax*, o que for menor. Caso a transmissão ocorra com sucesso, a estação reinicia o valor de CW para *aCWMin*.

3.4.1.4 DCF Básico

No DCF básico, uma estação pode enviar um quadro quando estiver em operação numa rede sem PC ou durante o CP de um BSS operando sob a PCF. A estação deve usar a função de sensoriamento do meio para determinar se este encontra-se ocioso por pelo menos um DIFS ou um EIFS, a depender do resultado da última transmissão. Caso o meio se encontre ocupado ou venha a se tornar ocupado no período de espaçamento entre quadros que precede o envio do quadro, a estação entra no procedimento de *backoff*. Caso o meio permaneça livre por todo o período de espaçamento, a estação irá enviar o quadro. No cabeçalho deste quadro, a estação deve indicar o tempo de reserva do meio, composto pelo tempo necessário para enviar o quadro somado a um SIFS e ao tempo necessário para a transmissão do quadro de recebimento (ACK). Este tempo é utilizado por todas as estações para atualizar os seus NAVs, indicando ao mecanismo de sensoriamento a reserva do meio.

Uma vez iniciada a transmissão, a estação continua o envio até ter enviado todo o quadro, mesmo que ocorra colisão, pois o protocolo considera que estações móveis são inábeis para detectar colisão, através da escuta do meio, e realizar simultaneamente a transmissão de sinais. Caso a transmissão seja efetuada com sucesso, a estação receptora deverá aguardar por um período igual a um SIFS e enviar um quadro de recebimento (ACK), informando no seu cabeçalho o tempo de transmissão do ACK para que as estações possam atualizar seus NAVs. Caso a transmissão falhe devido a uma colisão, a estação transmissora não irá receber nenhum quadro de confirmação e entrará no procedimento de *backoff*, tentando, posteriormente, a retransmissão do quadro que não teve a recepção confirmada.

Na figura 3.7, pode-se observar a seqüência de troca de quadros entre as estações envolvidas numa comunicação sob o DCF básico. É representada a atualização dos NAVs das demais estações, o que garante o adiamento das tentativas de acesso ao meio, ao menos por um período DIFS após o término da transmissão do quadro ACK.

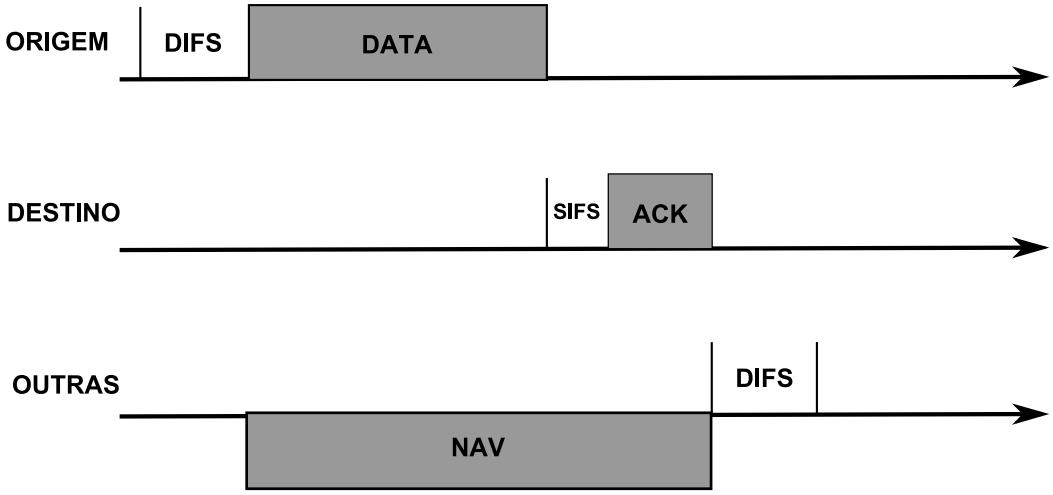

Figura 3.7 DCF Básico

3.4.1.5 DCF com RTS/CTS

As estações componentes de uma rede IEEE 802.11 são inábeis em detectar a ocorrência de colisão enquanto transmitem. Isto pode resultar em grande desperdício do uso do meio físico, em decorrência da continuidade de transmissão de longos quadros de dados mesmo na presença de uma colisão. Para evitar que isto aconteça, o protocolo prevê um mecanismo de reserva do meio, composto pela troca de dois quadros de alocação do canal: o RTS (*Request To Send*) e o CTS (*Clear To Send*). Os quadros de gerenciamento RTS e CTS são tipicamente menores que os quadros de dados. Assim, caso ocorra uma colisão no momento em que as estações estão tentando reservar o meio através da utilização do RTS/CTS, a perda no uso do meio tipicamente será menor.

Para alocar o meio, uma estação no modo DCF com RTS/CTS envia um quadro do tipo RTS à estação destino. Ao receber um RTS, a estação destino irá verificar se o seu NAV indica efetivamente que o meio está livre, enviando, neste caso, um CTS após aguardar um SIFS. Caso o NAV da estação destino indique que o meio se encontra ocupado, ela não enviará nenhum quadro. A estação origem irá aguardar pela chegada do quadro CTS por um período definido em *CTSTimeout*. Caso o quadro chegue dentro deste período, a estação origem irá aguardar um SIFS e então enviar os dados. Caso contrário, a estação irá assumir que houve uma colisão durante a transmissão do RTS e iniciará um procedimento de *backoff*. Após o envio dos dados, a estação destino deve aguardar por um novo SIFS e, finalmente, enviar um quadro de confirmação de recebimento (ACK), comportando-se as duas estações de modo similar ao definido para o DCF Básico. A figura 3.8 apresenta como se dá a troca de quadros e a atualização dos

NAVs das outras estações no modo DCF com uso de quadros RTS e CTS.

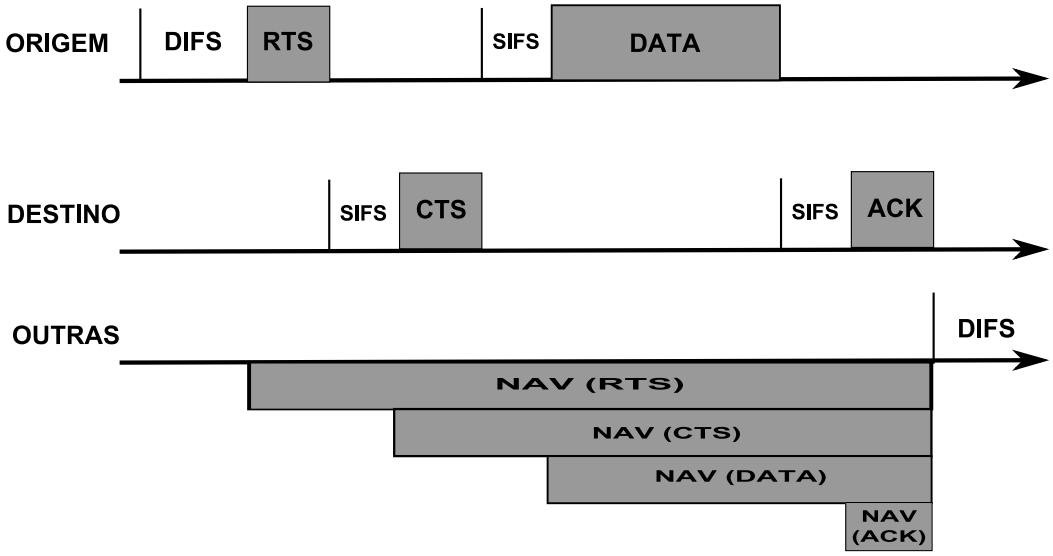

Figura 3.8 DCF com RTS/CTS

Conforme apresentado na figura 3.8, todos os quadros trocados entre as estações origem e destino carregam no seu cabeçalho a informação de quanto tempo o meio ainda estará sendo utilizado. Assim, as demais estações podem atualizar seus NAVs ao ler um quadro RTS, ou mesmo ao ouvir o quadro CTS enviado em resposta. Esta característica torna o mecanismo de RTS/CTS útil para tratar o problema do nó oculto. A figura 3.9 ilustra esta situação, considerando um BSS com três estações: a estação 2 é alcançável pelas estações 1 e 3, mas a estação 1 não está visível para a estação 3 e vice-versa. A estação 1 deseja enviar dados para a estação 2. Ao iniciar o processo de comunicação, a estação 1 enviará um RTS. Entretanto, a estação 3 não atualizará seu NAV, pois não leu o quadro enviado pela estação 1. Contudo, quando a estação 2 responder com o envio de um CTS, a estação 3 atualizará o seu NAV e a estação origem poderá continuar a transmissão de dados, sem que haja perigo de que a comunicação entre ela e a estação destino seja corrompida pelo envio de dados de uma estação que se encontra oculta para ela, mas alcançável pela estação destino.

O mecanismo de reserva do meio pelo uso dos quadros RTS/CTS representa mais um *overhead* na comunicação, à medida que insere mais troca de quadros que apenas servem para o gerenciamento e, portanto, não se constitui em informação útil. Assim, quanto menor for a quantidade de dados a ser enviada pela estação, maior será a quantidade de banda consumida por informação não útil. Em casos extremos, pode ocorrer que a quantidade de informação útil trocada seja menor que a quantidade de informação de controle utilizada para

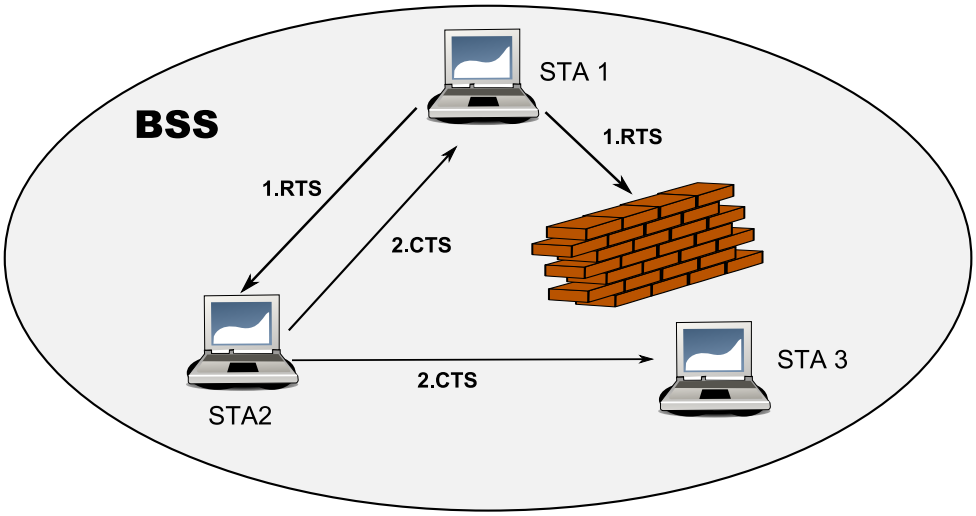

Figura 3.9 Usando o RTS/CTS para tratar o problema do nó oculto

este fim. Deste modo, o uso do mecanismo de RTS/CTS pode ser controlado através do parâmetro *dot11RTSThreshold*. Este parâmetro tem seu valor definido em cada estação e descreve se o mecanismo será sempre utilizado, nunca utilizado, ou utilizado apenas quando a estação precisar enviar quadros maiores que um determinado tamanho especificado.

3.4.2 PCF — Função de Coordenação Centralizada

A camada MAC da especificação IEEE 802.11 descreve adicionalmente um método de acesso denominado *Point Coordination Function* (PCF). A PCF é um método de acesso de implementação opcional, suportado apenas em redes com infra-estrutura. Na PCF, o acesso ao meio é controlado por uma estação específica, denominada PC (*Point Coordinator*). A função de PC é realizada pelo *Access Point* do BSS, entretanto não é obrigatório que todo AP esteja habilitado a operar como *point coordinator*.

Quando em operação no modo PCF, uma estação só irá transmitir caso tenha sido escalonada pelo PC ou esteja respondendo a uma transmissão prévia com um quadro de controle ACK. Assim, quando atua neste modo, diz-se que o protocolo encontra-se em um período livre de contenção (CFP — *Contention Free Period*).

A PCF possui suas regras de construção compatíveis com as regras especificadas pela DCF. Esta característica, associada à previsão de alternância entre as duas funções de coordenação, torna possível a coexistência entre estações que implementam e as que não implementam a PCF. Para o seu funcionamento, a PCF baseia-se na utilização do mecanismo virtual de sen-

soreamento do meio, que é fixado para um valor suficientemente alto por todos os quadros trocados na PCF. Além disso, um mecanismo de acesso prioritário ao meio, construído através do uso de um espaçamento entre quadros de menor duração (PIFS — *PCF Interframe Space*), é utilizado. Com isto, garante-se que mesmo estações que não implementam a PCF estejam sujeitas ao controle desta função de coordenação, quando se associam aos BSSs em que o AP atua como *Point Coordinator*.

Estações que não implementam a PCF podem ainda receber quadros que lhe sejam enviados durante o CFP. Entretanto, nem toda estação precisa estar preparada para responder a um escalonamento. Estações capazes de responder ao escalonamento de um PC (quadros do tipo CF-Poll) são chamadas de estações escalonáveis (*CF-Pollable*). Uma estação escalonável, quando escalonada por um PC, pode transmitir no máximo um quadro de dados (MPDU). Este quadro tanto pode ser destinado ao PC como a qualquer outra estação participante do BSS. Neste último caso, a estação destino irá indicar a recepção do quadro de dados através de um ACK. Caso uma falha aconteça e o ACK não seja transmitido/recebido, a estação não pode retransmitir o quadro de dados até que ela volte a ser escalonada pelo PC.

3.4.2.1 Início e Conclusão de um CFP

Para iniciar um CFP, o PC aguarda que o meio permaneça livre por um período (*Interframe Spacing*) denominado PIFS (*PCF Interframe Spacing*) e, em seguida, envia um quadro de controle chamado *beacon frame*. A duração de um PIFS é dada por $PIFS = aSIFSTime + aSlotTime$. Desta forma, o acesso pelo PC ao meio possui uma prioridade superior ao das outras estações, que necessitam aguardar um DIFS.

O *beacon frame* transporta no seu campo de informação de reserva de uso do meio um valor alto definido em CFPMaxDuration, de modo que para iniciar um CFP, as estações armazenam em seus NAVs um tempo de reserva do meio correspondente à duração de todo o período livre de contenção. Assim, após a transmissão do *beacon*, o protocolo ingressa em um período livre de contenção.

Para encerrar um período livre de contenção, o PC deve transmitir um quadro do grupo CF-End. Uma estação, ao detectar o envio de um CF-End, irá zerar o seu NAV, fazendo com que as regras de acesso da DCF sejam automaticamente adotadas.

3.4.2.2 Comportamento do Protocolo durante o CFP

Durante a duração do CFP, o PC possui o controle do uso do meio e pode temporariamente cedê-lo a outras estações, escalonando-as. No início do CFP, o PC deverá enviar um quadro

do tipo CF-Poll ou CF-Poll+Data a uma estação escalonável, de modo a transferir o controle e uso do meio a esta estação. No primeiro caso, o PC apenas escalona a estação, enquanto que no segundo caso, ele não só escalona a estação como lhe envia dados de modo combinado no mesmo quadro. Na seqüência, após a passagem de um SIFS, a estação poderá então transferir dados para o PC ou para uma outra estação qualquer, ou simplesmente retornar o controle do meio para o PC através do envio de um quadro Null, caso não tenha recebido dado, ou através de um quadro CF-ACK, caso ela tenha sido escalonada por um quadro CF-Poll+Data.

Caso a estação deseje transferir dados para uma outra estação que não o PC, o processo de controle somente estará completo após a recepção do quadro de confirmação ACK, remetido pela estação receptora após um período SIFS, ou, eventualmente, após a ocorrência de um *timeout*.

Caso os dados enviados pela estação se destinem ao próprio PC, ele aguardará o próximo ciclo para enviar a confirmação.

A cada novo ciclo, o PC poderá optar por transferir ou não o controle para uma estação. Caso o PC deseje escalonar uma estação deverá enviar um quadro da família CF-Poll, isolado ou combinado com outros quadros, para a estação a ser escalonada. Assim, se no ciclo anterior, o PC tiver recebido dados e estes estiverem pendentes de confirmação é necessário que no quadro enviado pelo PC, este insira uma confirmação do tipo CF-ACK. Ou seja, caso exista a necessidade de um quadro de confirmação, e o PC deseje escalonar uma nova estação, ele enviará um quadro do tipo CF-ACK+CF-Poll. A informação de confirmação será consumida pela estação que remeteu dados para o PC no ciclo anterior, enquanto que a informação de escalonamento será consumida pela estação a quem o quadro é endereçado. Caso o PC não deseje escalonar uma nova estação, o PC poderá apenas enviar um quadro CF-ACK.

Ainda a cada ciclo, o PC poderá remeter dados para uma das estações, segundo a mesma idéia de quadros combinados descrita acima para os quadros de confirmação. Deste modo, desejando o PC escalonar uma estação e destinar dados a ela, poderá fazê-lo através do envio de um quadro Data+CF-Poll. Adicionalmente, se o PC necessitar confirmar a recepção de um quadro de dados que lhe tenha sido destinado no ciclo anterior, o quadro a ser enviado deverá ser do tipo Data+CF-ACK+CF-Poll. Caso o envio de dados deva ocorrer sem que a transferência do controle do meio deva ser feita, o PC poderá se valer do quadro Data+CF-ACK ou apenas do envio de um quadro de dados. Isto dependerá da necessidade de confirmação da recepção de dados anteriormente destinados a ele.

Estações escalonadas devem, antes de enviar um MPDU para uma outra estação, verificar se o tempo restante do CFP é suficiente para o envio dos dados e a posterior recepção do quadro ACK. Caso o tempo seja insuficiente deverá limitar-se a enviar um quadro Null ou um quadro

CF-ACK, conforme seja o caso.

A qualquer tempo, limitado à duração do CFP, o PC poderá optar por terminar o CFP, através do envio de um quadro do tipo CF-End ou CF-ACK+CF-End, a depender da necessidade do envio de quadros de confirmação.

3.4.3 Alternância entre CFP e CP

A especificação IEEE 802.11 prevê a necessidade da operação conjunta das duas funções de controle de acesso ao meio dentro de um mesmo BSS. Assim, deve ocorrer uma alternância entre a ocorrência de um CP e a ocorrência de um CFP. O PC deve periodicamente iniciar um novo período livre de contenção, o que feito através do envio de um quadro *beacon* para iniciar um CFP com uma periodicidade dada através de um parâmetro de configuração da rede denominado taxa de repetição dos CFP (CFPRate — *contention free repetition rate*).

A duração do CFP deve ser controlada pelo PC, entretanto ela não deve exceder o valor dado pelo parâmetro de configuração da rede CFPMaxDuration. A qualquer tempo, o PC pode vir a terminar um CFP, em função da lista de estações a serem escalonadas ou das características do trafego durante o CFP. Caso o meio encontre-se ocupado no momento em que o PC deveria transmitir o quadro *beacon* para iniciar o CFP, o início do período livre de contenção será adiado, e o seu tamanho será diminuído do tempo pelo qual este adiamento aconteceu. A figura 3.10 apresenta um diagrama de como se dá a coexistência temporal dos dois modos de contenção.

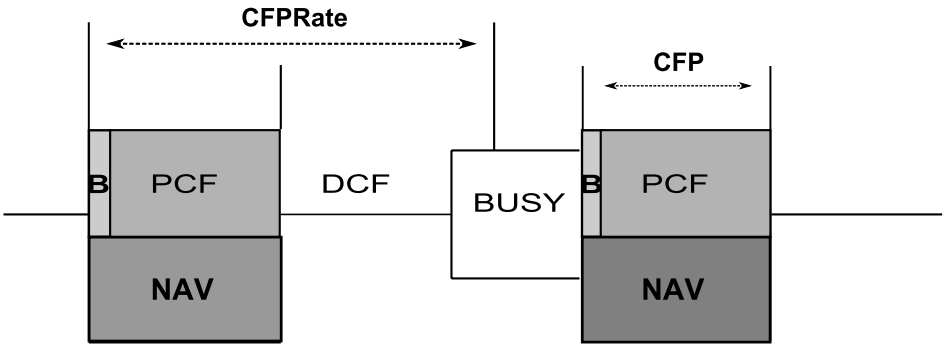

Figura 3.10 Coexistência temporal do dois períodos de contenção (CFP e CP)

Durante o CFP, o PC deverá estar sempre sensível ao tempo restante do CFP. Assim, o PC somente irá escalonar estações se o tempo restante for suficiente para a transmissão do quadro de escalonamento, para que a estação transmita ao menos o menor quadro de dados (suficiente para que a estação responda ao escalonamento) com um quadro do tipo CF-Ack ou Null) e para

a transmissão do quadro de encerramento do CFP (CF-End ou CF-Ack+CF-End).

Por sua vez, as estações, quando escalonadas, devem estar atentas ao tempo disponível de transmissão, de forma a não invadir o período reservado ao CP. Assim, a estação escalonada deve observar o tamanho do dado e se o tempo disponível comporta a sua transmissão. Em caso contrário, a estação deve se abster de transmitir o dado e enviar apenas um CF-Ack ou um quadro Null conforme seja o caso.

CAPÍTULO 4

UPPAAL

O corpo orgânico de cada vivente é uma espécie de máquina divina ou autômato natural que supera qualquer autômato artificial.

—GOTTFRIED LEIBNIZ (Cientista e Filósofo Alemão)

O UPPAAL (BLL+95) é uma ferramenta para modelagem, simulação e verificação de sistemas de tempo-real desenvolvida conjuntamente pelo BRICS (*Basic Research In Computer Science*) da *Aalborg University* e pelo *Department of Computer Systems* da *Uppsala University*. Esta ferramenta está disponível gratuitamente para fins não comerciais.

O UPPAAL baseia-se em algoritmos simbólicos e composicionais clássicos de verificação de modelos, e utiliza linguagens para especificação e modelagem capazes de tratar limites temporais. O modelo utilizado no UPPAAL para a especificação de sistemas é uma variação dos autômatos temporizados (*timed automata*) (ACD90). Por sua vez, a linguagem de especificação utilizada pela ferramenta é uma lógica de tempo denso denominada *Logic for Safety and Bounded Liveness Properties* ($\mathscr{L}_s$) (BLL+95). Esta lógica é um fragmento da lógica de tempo denso $\mathscr{L}_v$ (LLW95).

Este capítulo apresenta na seção 4.1 os autômatos temporizados. A seção 4.2 apresenta como estes autômatos podem ser compostos, para representar sistemas paralelos, concorrentes e distribuídos. O modo como o UPPAAL define seus sistemas através das redes de autômatos, além de algumas extensões de alto nível são exibidas na seção 4.3. Finalmente, a seção 4.4, exibe a linguagem $\mathscr{L}_s$ e mostra como propriedades de sistemas podem ser descritas nesta linguagem.

4.1 Autômatos temporizados

O modelo utilizado pelo UPPAAL para a especificação de sistemas é uma variação dos autômatos temporizados. Um autômato temporizado é, essencialmente, um autômato de estados finitos, composto por um conjunto de nós ligados por arestas, ao qual é associado um conjunto finito de relógios de tempo denso. Neste contexto, o termo relógio de tempo denso é utilizado

para denotar variáveis relógios que podem assumir valores reais. No UPPAAL, estes relógios são ainda definidos como ideais, ou seja, todos eles evoluem a uma taxa constante. Além das variáveis relógios, a extensão de autômatos temporizados utilizada no UPPAAL permite a adição de variáveis inteiras, booleanas e canais, além da declaração de arranjos e constantes numéricas.

Para restringir o comportamento do autômato, podem ser associadas às suas arestas um conjunto de expressões booleanas, denominadas guardas. Os guardas são utilizados para indicar se as arestas estão ou não habilitadas a acontecer. Entretanto, a existência de um guarda representa apenas uma condição de habilitação de uma aresta, não garantindo que a transição necessariamente venha a ocorrer. Deste modo, se houver apenas a utilização de guardas, um autômato pode permanecer indefinidamente num mesmo nó.

Algumas variações dos autômatos temporizados foram propostas visando garantir que os autômatos necessariamente apresentem algum progresso. Em 1990, Alur e Dill (AD90) introduziram condições de aceitação *Büchi* para tratar deste problema. Nesta variação de autômatos (*Timed Büchi*), alguns nós são marcados como nós de aceitação. A semântica definida para estes autômatos garante que, para todas as execuções válidas, um nó de aceitação deve ser visitado infinitas vezes. UPPAAL faz uso de uma forma mais intuitiva de garantia de progresso de um autômato temporizado. Na variante de autômatos temporizados utilizada pela ferramenta e denominada *Timed Safety Automata*(HNSY94), associam-se expressões booleanas aos nós, de forma a restringir a permanência do autômato em cada um deles, apenas enquanto tais expressões permaneçam satisfeitas. Estas expressões são denominadas invariantes.

Neste trabalho sempre que nos referirmos aos autômatos temporizados estaremos tratando de um *Timed Safety Automata*.

4.1.1 Declarações Locais e Globais

Em UPPAAL são utilizadas declarações para definir constantes, variáveis relógios, variáveis inteiras, variáveis inteiras com limites, variáveis booleanas, variáveis canais de sincronização e variáveis arranjos. As declarações podem ser locais a um autômato ou globais. São exemplos de declarações:

- `const TEMP_SET_POINT 100`; declara uma constante de nome `TEMP_SET_POINT` com valor igual a 100;
- `bool pressureOk,tempOk := true`; declara duas variáveis booleanas. A primeira de nome `pressureOk` e a segunda, de nome `tempOk`, iniciada com `true`;

- `int`$[1,12]$ `vOut` $= 12$; declara uma variável inteira de nome `vOut` com domínio entre 1 e 12 e a inicia com o valor 12. A qualquer tempo, uma atribuição fora da faixa irá levar o autômato a um erro de estado sucessor inválido;
- `int clpIn`$[2][2] := \{\{5,-5\},\{0,5\}\}$; declara uma arranjo bidimensional de inteiros de nome `clpIn` e o inicializa;
- `clock c_pumpTimer`; declara uma variável relógio de nome `c_pumpTimer`;
- `chan start`; declara uma variável canal de nome `start`.

4.1.2 Semântica Operacional

Um autômato temporizado é composto por um conjunto de nós e arestas que unem estes nós. A cada nó pode ser associada uma condição invariante e a cada aresta pode-se associar um guarda, de modo a se alterar o comportamento padrão do autômato.

O estado de um autômato temporizado pode ser representado através de um par (l,u), onde l é o nó onde o autômato atualmente se encontra e u é o conjunto de valores atualmente assumidos por cada uma de suas variáveis. O estado de um autômato temporizado evolui através de dois tipos de transições: a transição ação e a transição retardo.

Uma transição ação é definida como uma mudança de estado do autômato, que ocorre de modo instantâneo e que o leva de um nó a outro, através de uma aresta. A uma transição ação pode-se associar atribuições. Estas atribuições podem ser de valores reais não negativos, quando a variável envolvida for do tipo relógio, ou de valores inteiros, quando a variável envolvida for inteira. Em qualquer tempo, o valor de um relógio é dado pela soma do valor que lhe foi atribuído somado ao tempo decorrido desde então.

Uma transição retardo corresponde à simples passagem de tempo. Neste caso, o estado do autômato muda porque os valores dos relógios da rede de autômatos são modificados, embora o autômato se mantenha no mesmo nó em que se encontrava anteriormente.

De maneira informal, deve-se entender uma execução em um autômato temporizado como uma seqüência de transições, disparadas a partir do estado inicial l_0, que não violam os guardas e as invariantes associadas ao modelo expresso no autômato.

A próxima seção apresenta uma definição formal dos autômatos temporizados.

4.1.3 Definição Formal

Um relógio de tempo denso, representado através de uma variável relógio, é definido como uma variável capaz de assumir valores em $\mathbb{R}_+$ e cujo valor aumenta automaticamente com a passagem do tempo.

Definição 1 (Restrições Atômicas)**.** *Seja C um conjunto de relógios de tempo denso e I um conjunto de variáveis inteiras. Uma restrição atômica sobre os relógios de C é uma expressão da forma: $x \sim n$, onde $x \in C$, $\sim \in \{<, \leq, >, \geq, =\}$ e $n \in \mathbb{N}$.*

Uma restrição atômica sobre as variáveis inteiras I é uma expressão na forma: $i \sim n$, onde $i \in I$, $\sim \in \{<, \leq, >, \geq, =\}$ e $n \in \mathbb{Z}$.

Denomina-se por $\mathscr{C}_c(C)$ o conjunto de todas as restrições atômicas sobre relógios e por $\mathscr{C}_i(I)$ o conjunto de todas as restrições atômicas sobre I.

Definição 2 (Guardas e Invariantes)**.** *Seja C um conjunto de relógios de tempo denso e I um conjunto de variáveis inteiras. Um guarda g sobre C e I é uma fórmula gerada pela seguinte sintaxe: $g ::= r | g \,\&\&\, g$, onde $r \in (\mathscr{C}_c(C) \cup \mathscr{C}_i(I))$ e $\&\&$ denota uma conjunção. De modo similar, um invariante i sobre C e I é uma fórmula gerada pela seguinte sintaxe: $i ::= r | i \,\&\&\, i$, onde $c \in (\mathscr{C}_c(C) \cup \mathscr{C}_i(I))$ e $\&\&$ denota uma conjunção.*

Denomina-se por $\mathscr{G}(C,I)$, o conjunto de todos os guardas sobre C e I. E por $\mathscr{I}(C,I)$, o conjunto de todos os invariantes sobre C e I.

Definição 3 (Atribuições)**.** *Seja C um conjunto de relógios de tempo denso e I um conjunto de variáveis inteiras. Uma atribuição sobre C é uma tupla $\langle v, c \rangle$, onde $v \in C$ e $c \in \mathbb{N}$. Uma atribuição sobre I é uma tupla $\langle v, c \rangle$, que representa a atribuição $v = c$, onde $v \in I$ e $c \in \mathbb{Z}$.*

Denota-se $\mathscr{A}(C,I)$ para representar o conjunto de todas as atribuições possíveis sobre I e C.

Definição 4 (Autômato Temporizado)**.** *Um autômato temporizado A sobre um conjunto finito de ações Act, relógios C e um conjunto de variáveis inteiras I é uma tupla $A = \langle L, l_0, E, \mu \rangle$, onde:*

- *L é um conjunto finito de nós;*
- *$l_0 \in L$ é o nó inicial;*
- *$E \subseteq L \times \mathscr{G}(C,I) \times Act \times \mathscr{A}(C,I) \times L$ é o conjunto de arestas;*

- $\mu : L \rightarrow \mathscr{I}(C,I)$ *é uma função que associa invariantes aos nós.*

Alternativamente, pode-se escrever $l \overset{g,a,r}{\rightarrow} l'$ para representar uma aresta de l para l' com guarda g (condição de habilitação da transição), ação a e um conjunto de atribuições r, que devem ocorrer no instante da transição, caso $\langle l,g,a,r,l' \rangle \in E$.

A cada nó l, associa-se uma condição invariante $\mu(l)$. Esta condição é uma restrição sobre as variáveis do sistema, que deve ser satisfeita em qualquer instante da execução, caso o sistema se encontre no nó l.

Definição 5 (Estado de um Autômato Temporizado)**.** *O estado de um autômato temporizado A é denotado por um um par (l,u) onde $l \in L$ e u é uma atribuição que mapeia cada relógio em C para um valor em $\mathbb{R}^+$ e cada variável em I para um valor em $\mathbb{Z}$.*

A expressão $g(u)$ denota a atribuição u que satisfaz o guarda g. O estado inicial de A é (l_0,u_0), onde u_0 é a atribuição que mapeia todas as variáveis para 0.

Definição 6 (Transição Retardo)**.** *Seja (l,u) e (l',u') dois estados do autômato temporizado A, e seja $d \in R_*^+$. Então, uma transição retardo $\overset{\varepsilon(d)}{\rightsquigarrow}$ é definida como:*

$$(l,u) \overset{\varepsilon(d)}{\rightsquigarrow} (l',u') \iff \begin{cases} l' = l \\ u'(x) = u(x) & se\ x \in I \\ u'(x) = u(x) + d & se\ x \in C \\ u'(x) \in \mu(l') \end{cases}$$

Definição 7 (Transição Ação)**.** *Seja (l,u) e (l',u') dois estados do autômato temporizado A. Então, uma transição ação $\overset{g,a,r}{\rightsquigarrow}$ é definida como:*

$$(l,u) \overset{g,a,r}{\rightsquigarrow} (l',u') \iff l \overset{g,a,r}{\rightarrow} l' \wedge g(u) \wedge u'(x) \in \mu(l')$$

onde,

$$u'(x) = \begin{cases} c_0 & se\ x \in C \wedge \langle x,c_0 \rangle \in r \\ c_1 & se\ x \in I \wedge \langle x,c_1 \rangle \in r \\ u(x) & caso\ contrário \end{cases}$$

Definição 8 (Execução)**.** *Uma execução ξ sobre um autômato temporizado A é uma seqüência, possivelmente infinita, de estados do autômato:*

$$\xi = \langle (l_0,u_0),(l_1,u_1),(l_2,u_2),(l_3,u_3)... \rangle$$

onde, para migrar de um estado i para um estado $i+1$, o autômato deverá efetuar uma transição ação ou uma transição retardo.

Desta forma, em uma execução, o autômato inicia no nó l_0 com todas as variáveis com valor 0. A partir daí os relógios incrementam sincronamente no tempo no nó l, enquanto a invariante $I(l)$ permanecer satisfeita. A qualquer momento o autômato poderá seguir uma aresta $l \xrightarrow{g,a,r} l'$, desde que os relógios satisfaçam o guarda g. Quando esta transição ocorrer, acontecerão as atribuições em r e a ação a será realizada.

4.1.4 Representação Gráfica e Exemplo

Em UPPAAL, os autômatos podem ser representados graficamente: os nós são representados por círculos; o nó inicial por dois círculos concêntricos; e as arestas por setas, interligando os nós.

Na figura 4.1, é representado um modelo de funcionamento de uma bomba temporizada simples. O modelo é um autômato que opera sobre o conjunto de ações $Act = \{\varepsilon\}$, sobre o conjunto de relógios $C = \{\texttt{c\_pump}\}$ e sobre o conjunto de variáveis inteiras $I = \varnothing$. O símbolo ε indica uma ação vazia.

Este modelo é composto pelo conjunto de nós $L = \{\texttt{Off}, \texttt{On}\}$, pelo nó inicial $l_0 = \texttt{Off}$, pelo conjunto de arestas $E = \{(\texttt{Off}, \texttt{c\_pump} \geq 10, \varepsilon, \{\texttt{c\_pump} := 0\}, \texttt{On}), (\texttt{On},$
$\texttt{c\_pump} == 20, \varepsilon, \{\texttt{c\_pump} := 0\}, \texttt{Off})\}$, e pela função de associação de invariantes
$\mu = \{(\texttt{Off}, \texttt{c\_pump} \leq 15), (\texttt{On}, \texttt{c\_pump} \leq 20)\}$.

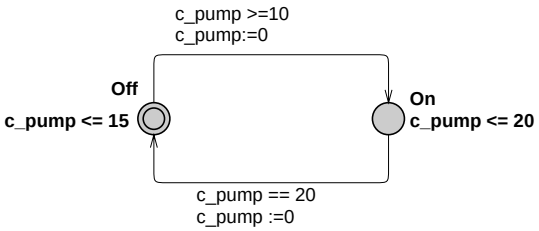

Figura 4.1 Modelo de uma Bomba Temporizada Simples

A bomba modelada possui um funcionamento cíclico. Esta bomba encontra-se inicialmente desligada e em algum momento entre o instante $t = 10$ e $t = 15$ ela começa a funcionar. Então, a bomba permanece acionada por exatamente 20 unidades de tempo, desligando-se em seguida e retornando ao início do ciclo.

Os nós `Off` e `On` foram utilizados, respectivamente, para representar a bomba desligada e a bomba em funcionamento. O guarda $\texttt{c\_pump} \geq 10$ da aresta $(\texttt{Off}, \texttt{c\_pump} \geq 10, \varepsilon,$

$\{\texttt{c\_pump} := 0\}, \texttt{On})$ indica que para ela estar habilitada e, por conseqüência, para que o autômato possa migrar do nó `Off` para `On` é necessário que o relógio `c_pump` esteja com um valor superior ou igual a 10, ou seja, que já tenha decorrido ao menos 10 unidades de tempo desde a última vez em que o relógio tenha sido reiniciado. Quando a transição ação associada a esta aresta acontecer, o relógio `c_pump` será reiniciado, conforme descreve o conjunto de atribuições $r = \{\langle \texttt{c\_pump}, 0\rangle\}$ desta aresta. Esta aresta é insuficiente, entretanto, para garantir que o autômato irá deixar o nó `Off` em algum momento. Para modelar o comportamento desejado, um invariante $i_{Off} = \texttt{c\_pump} \leq 15$ é associado ao nó `Off`. Assim, o autômato deve deixar este nó antes do relógio `c_pump` atingir valor superior a 15. Deste modo, o autômato migrará do nó `Off` para o nó `On`, em algum instante no intervalo $[10, 15]$.

Uma vez no nó `On`, o autômato pode permanecer por até 20 unidades de tempo, situação descrita pela existência do invariante $i_{On} = \texttt{c\_pump} \leq 20$ associado a este nó. Entretanto, a bomba não pode transitar para o nó `Off` antes de 20 unidades de tempo, pois a aresta somente estará habilitada quando se passarem 20 unidades de tempo de permanência do autômato no nó `On`, conforme restrição apresentada no guarda $\texttt{c\_pump} == 20$. Deste modo, a transição entre `On` e `Off` ocorre exatamente 20 unidades de tempo após a o autômato alcançar o nó `On`.

Uma execução possível deste autômato é $\xi = \langle (\texttt{Off}, \{\texttt{c\_pump} = 0.0\}), (\texttt{Off}, \{\texttt{c\_pump} = 0.5\}), (\texttt{Off}, \{\texttt{c\_pump} = 13.5\}), (\texttt{On}, \{\texttt{c\_pump} = 0.0\}), (\texttt{On}, \{\texttt{c\_pump} = 20.0\}), (\texttt{Off}, \{\texttt{c\_pump} = 0.0\}) ...\rangle$. Esta execução exibe o funcionamento de uma bomba que inicia desligada e em algum instante entre o tempo 13,5 e 15 é ligada, permanecendo assim por 20 unidades de tempo, quando então volta a ficar desligada.

4.2 Redes de Autômatos Temporizados

Para modelar sistemas que envolvam concorrência e sincronização, os autômatos temporizados podem ser incrementados com a noção de composição paralela, de modo a ser possível gerar redes de autômatos temporizados. Uma rede de autômato temporizado é uma composição paralela $A_1|...|A_n$ de um conjunto de autômatos temporizados A_1, ...,A_n combinados em um único sistema, através da utilização de uma função de sincronização (BY04). Este novo sistema é formado pelo produto cartesiano dos estados dos autômatos componentes e por uma função de transição que define como se processa a evolução do autômato resultante.

Na variante de autômatos temporizados apresentada pelo UPPAAL é utilizado um operador de composição paralela similar ao definido no CSP (Hoa85). Este operador define a possibilidade de intercalação entre as transições ação dos autômatos, assim como introduz a noção de

co-ação. A co-ação modela a possibilidade de sincronização entre duas transições ação através da comunicação via portas complementares. Neste último caso, os dois autômatos evoluem através de suas arestas ao mesmo tempo.

Na modelagem da comunicação síncrona entre dois autômatos o alfabeto de ações *Act* é ampliado para incluir símbolos de ações internas τ e símbolos de co-ações α. As co-ações são ações que pertencem ao alfabeto de mais de um autômato e que modelam evoluções síncronas entre dois autômatos.

Para a representação de uma co-ação são utilizados $\alpha?$ e $\alpha!$ que representam a sincronização entre dois autômatos sobre o canal α. O símbolo $\alpha?$ representa uma operação de leitura sobre o canal α. O símbolo $\alpha!$ representa uma operação de escrita sobre este mesmo canal. As operações de leitura e escrita são ditas complementares entre si.

Um par de sincronização será escolhido de modo não determinista se várias co-ações possíveis estiverem habilitadas.

4.2.1 Semântica Operacional

A semântica de uma rede de autômatos temporizados é apresentada de forma similar àquela adotada para um único autômato, compreendido como um sistema temporizado de transição. Assim, o estado de uma rede de autômatos é uma par $\langle l, u \rangle$, onde l é um vetor dos nós correntes de cada um dos autômatos que compõem a rede e u é um conjunto dos valores assumidos por cada uma das variáveis do sistema no instante de tempo atual.

A rede de autômatos evolui através de dois tipos de transições: transições retardo e transições ação. Uma transição retardo ocorre de modo similar àquele definido para a transição retardo de um único autômato temporizado, considerando-se que a invariante do vetor de nós l é dado pela conjunção de todas as invariantes dos nós que dele fazem parte. Para as transições ação, duas regras são adotadas: caso a ação a ser disparada seja uma ação interna τ, apenas um dos autômatos realizará a transição; caso a ação seja uma co-ação α, representando uma operação de sincronização entre dois autômatos sobre o canal α, os dois autômatos envolvidos progridem simultaneamente.

4.2.2 Definição Formal

Definição 9 (Função de Sincronização)**.** *Seja* $\Phi \subseteq Act \times Act$ *uma função parcial tal que:*

$$(\alpha_i, \alpha_j) \in \Phi \Rightarrow (\alpha_j, \alpha_i) \in \Phi, \textit{para todo } \alpha_i, \alpha_j$$

Definição 10 (Composição Paralela)**.** *Sejam* $A_1 = \langle L_1, l_{1,0}, E_1, \mu_1 \rangle$, $A_2 = \langle L_2, l_{2,0}, E_2, \mu_2 \rangle$ *dois autômatos temporizados. Então a composição paralela* $A_1|A_2$ *é um autômato temporizado* $A = \langle L, l_0, E, \mu \rangle$ *onde:*

- $(l_1 \times l_2) \in L$*, qualquer que seja* $l_1 \in L_1$ *e* $l_2 \in L_2$*;*
- $l_0 = (l_{1,0} \times l_{2,0})$*;*
- E *é definida como se segue:*

$$e \in E \iff e = \begin{cases} (l_1|l_2) \overset{g,a,r}{\rightarrow} (l_1'|l_2') & se\ (l_1 \overset{g_1,a_1,r_1}{\rightarrow} l_1') \wedge (l_2 \overset{g_2,a_2,r_2}{\rightarrow} l_2') \wedge (g = g_1 \wedge g_2) \\ & \wedge((a1,a2) \in \Phi) \wedge (r = r_1 \cup r_2) \\ (l_1|l_2) \overset{g,a,r}{\rightarrow} (l_1'|l_2) & se\ l_1 \overset{g,a,r}{\rightarrow} l_1' \\ (l_1|l_2) \overset{g,a,r}{\rightarrow} (l_1|l_2') & se\ l_2 \overset{g,a,r}{\rightarrow} l_2' \end{cases}$$

- $\mu(l_1|l_2) = \mu(l_1) \bigwedge \mu(l_2)$*;*

A composição paralela é uma operação comutativa e associativa.

4.2.3 Representação Gráfica

Um exemplo de uma rede de autômatos composta por dois autômatos temporizados é apresentado na figura 4.2. Esta rede de autômatos tem por base a bomba temporizada apresentada na figura 4.1.

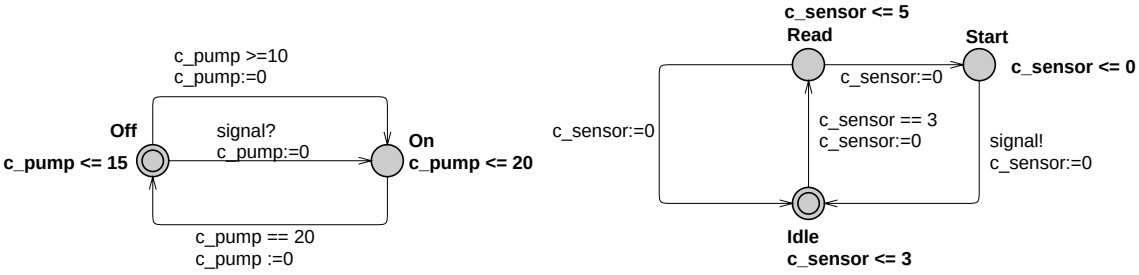

Figura 4.2 Modelo de uma Bomba Temporizada com Sensor

Neste novo modelo, a bomba continua sendo disparada periodicamente, como no modelo anterior. Entretanto, ela pode ainda ser disparada pela ocorrência de um evento que é detectado por um sensor (representado pelo autômato à direita). O sensor funciona após três unidades de tempo de inatividade (nó `Idle`) e adquire o valor de uma variável de ambiente controlada (nó

`Read`). O processo de aquisição, composto da leitura efetiva seguido do processamento do valor encontrado, dura no máximo 5 unidades de tempo. Em seguida, o sensor pode optar por voltar ao estado de inatividade, caso a variável controlada esteja dentro dos valores esperados. Caso o valor da variável não seja o esperado, o sensor irá enviar um sinal, representado pelo canal `signal`! para a ativação imediata da bomba (esta opção é representada através de uma escolha não determinista no autômato). O sinal de ativação é detectado pela bomba desligada (nó `Off`), que irá ativar a transição com ação `signal`?. Então, a bomba será ligada, o que é modelado pelo movimento do autômato para o nó `On` através da aresta que descreve esta transição ação. A partir daí a bomba comporta-se como o modelo anterior, permanecendo ligada por 20 unidades de tempo e desligando-se ao final deste período.

Uma possível execução parcial da rede de autômatos temporizado apresentada, é dada a seguir:

$$\begin{aligned}\xi = \langle(&\{\texttt{Off},\texttt{Idle}\},\{\texttt{c\_pump} = 0.0, \texttt{c\_sensor} = 0.0\}),\\ &(\{\texttt{Off},\texttt{Idle}\},\{\texttt{c\_pump} = 2.5, \texttt{c\_sensor} = 2.5\}),\\ &(\{\texttt{Off},\texttt{Read}\},\{\texttt{c\_pump} = 3.0, \texttt{c\_sensor} = 0.0\}),\\ &(\{\texttt{Off},\texttt{Start}\},\{\texttt{c\_pump} = 3.2, \texttt{c\_sensor} = 0.2\}),\\ &(\{\texttt{On},\texttt{Idle}\},\{\texttt{c\_pump} = 0.0, \texttt{c\_sensor} = 0.0\}), \ldots \rangle\end{aligned}$$

Esta execução descreve que o sensor realiza a leitura após três unidades de tempo de inatividade, e, após 0,2 unidades de tempo após a leitura, completa o processamento do valor lido e opta por disparar a bomba. A bomba é imediatamente disparada e o sensor retorna a um período de inatividade.

4.3 Modelagem de Sistemas em UPPAAL

Um sistema é representado no UPPAAL, essencialmente, por uma rede de autômatos temporizados. Para a definição de sistemas, UPPAAL faz uso de uma hierarquia composta de processos e modelos, conforme descrito nesta seção e esquematizado na figura 4.3.

Um modelo em UPPAAL é descrito em termos de autômatos temporizados incrementado com variáveis. Este autômato pode ser parametrizado através da utilização de variáveis ou constantes. Desta forma, um mesmo autômato pode se comportar de modo particular em função do valor assumido pelas variáveis parâmetros, no momento da instanciação do processo. A declaração dos parâmetros segue a mesma sintaxe da declaração de variáveis, descrita na seção 4.1.1. Por exemplo, a declaração `PUMP(chan signal)` indica a declaração de um modelo de nome `PUMP` que possui uma variável parâmetro do tipo canal e de nome `signal`.

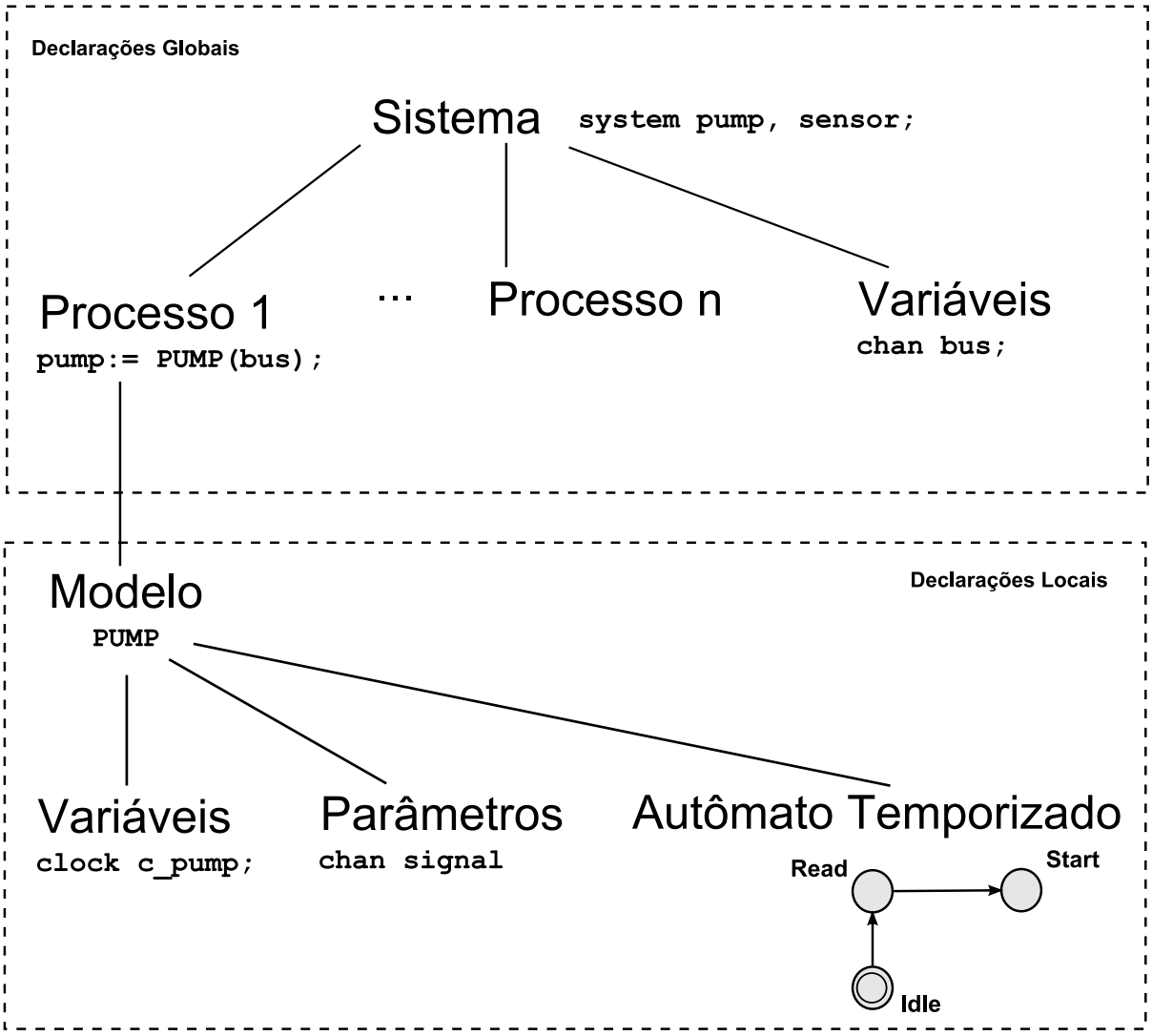

Figura 4.3 Modelo da Hierarquia de um Sistema no UPPAAL

A partir de cada modelo pode-se criar uma ou mais instâncias, que são denominadas processos. Na instanciação dos processos, são fornecidos os valores dos parâmetros para os modelos, caso tais parâmetros existam. Cada instância de modelo é atribuída a uma variável processo como na declaração `pump := PUMP(bus);`. Neste exemplo, o processo `pump` é uma instância do modelo `PUMP` parametrizado através da variável `bus`. Caso o modelo não possua parâmetro, a lista de parâmetros deverá ser mantida vazia. São exemplos válidos de declaração de processos:

- `sensor:=SENSOR();`

- `control := P(x,1);`
- `luz3Estados := LUZ3ESTADOS(4,100);`

Finalmente, a definição de sistemas permite a composição em paralelo de processos previamente criados para formar uma rede de autômatos temporizados. Para isto é usada a declaração `system` seguida de uma lista de identificadores de processo. Por exemplo, um sistema composto de dois processos de nome `pump` e `sensor` é declarado através da definição de sistema `system pump, sensor;`.

4.3.1 Extensões de alto nível

Visando facilitar o processo de modelagem, UPPAAL fornece uma série de extensões de alto nível. Estas extensões podem ser construídas com os elementos básicos dos autômatos temporizados. Sua adição, entretanto, permite maior facilidade e clareza na modelagem. Na versão atual do UPPAAL as principais extensões são: os canais *urgent* e *broadcast* e os nós *urgent* e *commited*.

4.3.1.1 Canais *urgent*

Os canais *urgent* são utilizados para modelar transições síncronas urgentes. Transições síncronas urgentes são aquelas que devem ocorrer logo que elas estejam habilitadas. Deste modo, quando uma transição urgente está habilitada, nenhuma transição retardo poderá acontecer. Entretanto, intercalações com outras transições ação, que não consomem tempo, ainda são possíveis. Uma transição que possua um canal urgente para sincronização não permite a adoção de variáveis relógios na expressão que forma o seu guarda. Canais urgentes são declarados através do modificador `urgent` aplicado à variável canal. Por exemplo, `urgent chan fire;` declara um canal *urgent* de nome `fire`.

4.3.1.2 Canais *broadcast*

Canais *broadcast* são utilizados para modelar sincronizações do tipo 1 para n. Nesta espécie de canal, caso uma transição possua uma ação de sincronização `canal`!, todas as transições que possuam ação de sincronização `canal`? e que estejam habilitadas a ocorrer, irão ocorrer. Entretanto, a ação de sincronização `canal`! não depende da existência de uma ação de sincronização `canal`? para acontecer, ou seja, a operação de escrita sobre canais *broadcast* é uma operação não bloqueante. Guardas com expressões sobre relógios não podem ser utili-

zados nas transições de leitura sobre canais *broadcast*. Com relação às ações de atualização em canais urgentes, a primeira a ser executada é a que pertence à transição de escrita sobre o canal. Em seguida, as atualizações são executadas nas transições de leitura, na mesma ordem em que os seus processos aparecem na definição do sistema (cláusula `system`). A declaração de canais *broadcast* é feita através do modificador `broad` aplicado à variável canal. Assim `broad chan update;`, declara uma variável de nome `update` do tipo canal *broadcast*.

4.3.1.3 Nó *urgent*

Os nós *urgent* representam nós no autômato em que não é possível a ocorrência de transições retardo, ou seja, o tempo não pode passar enquanto o autômato encontra-se neste nó. Os nós urgentes são semanticamente equivalentes a se adicionar uma variável relógio extra (`clock x;`) ao autômato, reiniciá-la em todos as arestas que alcançam o nó (`x:=0`), e adicionar um invariante `x<=0` neste mesmo nó.

Entretanto, nós *urgent* permitem a ocorrência de intercalações. Assim, é possível que o autômato permaneça em um nó deste tipo enquanto transições ações de outros autômatos da rede acontecem. Nós *urgent* são representados graficamente através de uma letra U dentro do círculo que representa o nó.

4.3.1.4 Nó *commited*

Os nós *commited* são de execução ainda mais restritiva que a dos nós *urgents*. Quando um autômato encontra-se em um nó *commited*, as únicas transições que podem ser disparadas são as transições ação que partem destes nós.

Diz-se que o estado da rede de autômatos é *commited*, se ao menos um dos nós deste estado for *commited*. Neste caso, a rede não suportará transições retardo e a próxima transição deve necessariamente envolver a saída de pelo menos um dos nós *commited*, ou seja, a rede também não aceitará intercalações com transições ação de autômatos da rede, que não envolvam a partida de um nó *commited*. Para representar graficamente nós *commited*, estes são marcados com uma letra C dentro do círculo que representa o nó.

Como exemplo do uso das extensões de alto nível, a figura 4.4 exibe o modelo do sensor do conjunto Bomba Temporizada/Sensor modificado. Esta modificação representa através de um nó *commited* o acionamento imediato da bomba pelo sensor. O funcionamento deste modelo é idêntico àquele apresentado na figura 4.2, onde o controle de permanência no nó `Start` é realizado com o uso de invariantes e guardas que consideram o valor assumido pelo relógio `c_sensor`.

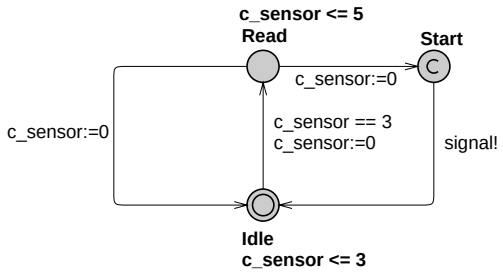

Figura 4.4 Modelo do Sensor do Conjunto Bomba/Sensor com uso de nó *commited*

4.4 Lógicas Temporizadas e a Linguagem de Especificação do UPPAAL

Dado um sistema modelado através de uma rede de autômatos, deseja-se verificar se determinadas propriedades são satisfeitas. Algumas destas propriedades são simplesmente temporais, no sentido de que não tratam de limites de tempo, mas, ao invés disto, apenas descrevem a ordem em que determinados eventos devem ocorrer. Para estas propriedades é suficiente verificar se um determinado estado desejado será futuramente alcançado, ou que estados de erro nunca acontecerão. Outros tipos de propriedades, entretanto, necessitam manipular explicitamente limites temporais. Neste caso deseja-se tratar da quantidade de tempo que se passa entre a ocorrência de dois eventos. Estas propriedades são chamadas de propriedades de tempo-real. Uma opção para representar adequadamente as propriedades de tempo-real é fazer uso das chamadas lógicas temporizadas. Estas lógicas são extensões das lógicas temporais que podem expressar condições de retardos máximos através da associação de condições booleanas aos operadores temporais.

O verificador de modelos do UPPAAL é projetado para verificar as fórmulas de um subconjunto de uma lógica temporizada (TCTL) estendida à partir da CTL (*Computational Tree Logic*). Esta lógica denomina-se *Logic for Safety and Bounded Liveness Properties* ($\mathscr{L}_s$) e de modo similar à CTL, suas fórmulas são formadas por fórmulas de estado e de fórmulas de caminho. Fórmulas de estado são expressões que podem ser avaliadas sem nenhuma preocupação com o comportamento do modelo, necessitando serem verificadas apenas no estado. Por exemplo, para avaliar a expressão `x==0`, basta apenas se verificar se o valor da variável `x` é igual a `0` no estado corrente do autômato. Fórmulas de caminho, por sua vez, implicam em raciocínio sobre o comportamento do modelo dentro de uma determinada execução computacional válida. Diferentemente de CTL, $\mathscr{L}_s$ não permite o aninhamento de fórmulas de estado.

As fórmulas de estado em $\mathscr{L}_s$ são formadas segundo as seguintes regras:

- Um predicado atômico é uma fórmula de estado. São predicados atômicos:
  - $P_i.S_j$ onde P_i é um processo do sistema e S_j é um nó deste processo. O predicado é verdade se e somente se P_i encontra-se no nó S_j;
  - $v_i \prec n$ onde v_i é uma variável inteira, relógio ou booleana, $\prec \in \{\leq, \geq, =\}$ e $n \in \mathbb{N}$. O predicado é verdade se o valor de v_i relaciona-se com n através de $\prec$;
  - *deadlock*. O predicado é verdade se e somente se para todas as transições existentes não existe nenhum estado sucessor válido, ou seja que não viole os guardas e invariantes do sistema.
- se ϕ e ψ são fórmulas de estado então, as fórmula abaixo são fórmulas de estado:
  - $\phi \wedge \psi$. A fórmula é verdadeira se e só se a fórmula ϕ e a fórmula ψ são verdadeiras;
  - $\phi \vee \psi$. A fórmula é verdadeira se e só se a fórmula ϕ ou a fórmula ψ são verdadeiras;
  - $\phi \Rightarrow \psi$. A fórmula é verdadeira se e só se caso a fórmula ϕ seja verdadeira então ψ também será verdadeira;
  - $\neg\phi$. A fórmula é verdadeira se e só se a fórmula ϕ é falsa.

Assumindo ϕ e ψ como fórmulas de estado, são fórmulas de caminho:

- $\mathtt{A}\Box\phi$ - Invariavelmente ϕ
- $\mathtt{E}\Diamond\phi$ - Possivelmente ϕ
- $\mathtt{A}\Diamond\phi$ - Sempre Futuramente ϕ
- $\mathtt{E}\Box\phi$ - Potencialmente Sempre ϕ
- $\phi \rightarrow \psi$ - ϕ sempre leva a ψ, ou seja, $\mathtt{A}\Box(\phi \Rightarrow \mathtt{A}\Diamond\psi)$

A tabela 4.1 apresenta as fórmulas válidas para $\mathscr{L}_s$, conforme descrito, e a sintaxe equivalente destas fórmulas no UPPAAL.

A partir de uma rede de autômatos é possível gerar uma árvore computacional infinita. Um ramo desta árvore é composto por uma seqüência de estados, que representa um caminho computacional de execução possível. A semântica dos operadores de ($\mathscr{L}_s$) sobre esta árvore é definida da seguinte forma:

Fórmula Lógica	Sintaxe no UPPAAL ($\mathscr{L}_s$)
$\mathtt{A}\Box\phi$	`A[]` ϕ
$\mathtt{A}\Diamond\phi$	`A<>` ϕ
$\mathtt{E}\Box\phi$	`E[]` ϕ
$\mathtt{E}\Diamond\phi$	`E<>` ϕ
$\phi \rightarrow \psi$	ϕ `-->` ψ
$\phi \wedge \psi$	ϕ `&&` ψ ou ϕ `and` ψ
$\phi \vee \psi$	ϕ `||` ψ ou ϕ `or` ψ
$\phi \Rightarrow \psi$	ϕ `imply` ψ
$\neg\phi$	`not`ϕ
$P_i.S_j$	P_i `.` S_j
$v_i \leq n$	v_i `<=` n
$v_i \geq n$	v_i `>=` n
$v_i = n$	v_i `==` n
deadlock	`deadlock`

Tabela 4.1 Sintaxe das fórmulas aceitas pelo UPPAAL

- as letras `A` e `E` são quantificadores de caminho. `A` é usada para denotar que a propriedade é verdadeira para todos os caminhos computacionais, enquanto `E` é usada para denotar que a propriedade é verdadeira em pelo menos um caminho computacional,
- os símbolos $\Box$ e $\Diamond$ são operadores temporais. $\Box$ é utilizado para denotar que todos os estados em um caminho satisfazem a propriedade, enquanto $\Diamond$ denota que pelo menos um estado no caminho satisfaz a propriedade.

4.4.1 Classes de Propriedades Verificadas no UPPAAL

O processo de verificação de modelos se baseia na descrição das propriedades do sistema em termos de fórmulas da lógica aceita pelo verificador e a posterior submissão destas fórmulas ao mecanismo de verificação da ferramenta. Embora reconhecidamente imprecisa, mas útil do ponto de vista didático, a classificação a seguir relaciona as propriedades de um sistema computacional em cinco grupos (BFB01):

- propriedades de alcançabilidade (*reachability*);
- propriedades de segurança (*safety*);
- propriedades de animação (*liveness*);

- propriedade de ausência de deadlock (*deadlock freeness*);
- propriedades de justiça (*fairness*).

4.4.1.1 Propriedades de Alcançabilidade

As propriedades de alcançabilidade são tipicamente as propriedades mais simples de serem verificadas. Em seu formato mais básico, uma propriedade de alcançabilidade indica se é possível se atingir um determinado estado do autômato a partir do estado inicial, após um número finito de transições, ou seja, que uma dada fórmula de estado poderá ser eventualmente satisfeita a partir do estado inicial do modelo.

As propriedades de alcançabilidade são úteis por expressar a possibilidade de se alcançar estados desejáveis dos modelos. Embora tais garantias sejam garantias fracas e insuficientes para expressar correção, elas descrevem e validam o comportamento básico do modelo, retirando a sua trivialidade.

Em $\mathscr{L}_s$, propriedades de alcançabilidade são representadas através de fórmulas do tipo $\mathtt{E}\Diamond\phi$, onde ϕ é uma fórmula de estado.

4.4.1.2 Propriedades de Segurança

Uma propriedade de segurança descreve que, sob certas circunstâncias, um determinado evento nunca ocorre. Deste modo, as propriedades de segurança são utilizadas para expressar que algo ruim ou indesejado nunca irá acontecer. Esta classe de propriedades normalmente descreve condições invariantes do sistema.

Um modo de expressar tais propriedades é através da negação de uma fórmula de estado de uma propriedade de alcançabilidade ($\mathtt{E}\Diamond\mathtt{not}\phi$.). Neste caso, o que se está afirmando é que um estado indesejável, descrito pela propriedade ϕ, nunca será alcançado.

Entretanto, uma forma mais natural de se expressar esta espécie de propriedade é formulá-la positivamente, ou seja, garantindo que algo desejável é invarialvelmente verdade. Em $\mathscr{L}_s$, esta representação é realizada através de fórmulas do tipo $\mathtt{A}\Box\phi$, onde ϕ é uma fórmula de estado.

4.4.1.3 Propriedades de Animação

Uma propriedade de animação descreve o fato de que, sob certas circunstâncias, determinado evento irá ocorrer em algum momento.

Deve-se observar que as propriedades de animação não se confundem com as propriedades de alcançabilidade. Propriedades de alcançabilidade expressam que determinados estados

podem ser alcançados. Entretanto, não existe a garantia de que, necessariamente, tais estados venham a ocorrer. Deste modo, propriedades de animação definem propriedades mais fortes, na medida que garantem que algo irá eventualmente acontecer e não apenas que pode vir a acontecer.

Em $\mathscr{L}_s$, representa-se a forma mais simples de uma propriedade de animação através da fórmula $\mathtt{A}\Diamond\phi$, onde ϕ é uma fórmula de estado. Esta fórmula indica que o estado ϕ será futuramente alcançado para todos os caminhos de execução possíveis.

Uma forma mais complexa de propriedade de animação é aquela que estabelece que a ocorrência de um estado necessariamente leva a ocorrência de um outro, ou seja, que para qualquer execução, se um evento acontecer, necessariamente se seguirá futuramente a ocorrência de outro evento. Para representar este tipo de propriedade, é necessário se aninhar operadores de caminho numa fórmula do tipo $\mathtt{A}\Box(\phi \Rightarrow \mathtt{A}\Diamond\psi)$. Como o sub-conjunto de CTL abrangido por $\mathscr{L}_s$ não suporta esta espécie de fórmula, $\mathscr{L}_s$ inclui a fórmula $\phi \rightarrow \psi$ para representar esta classe de propriedades.

Finalmente, quando as propriedades de *liveness* envolvem relógios, ou seja, além de se verificar se algum estado será futuramente alcançado, deseja-se verificar qual o máximo retardo ocorrido, elas são conhecidas por *bounded-liveness* ou de temporalidade (*timeliness*). Do ponto de vista teórico, as propriedades de temporalidade são, na verdade, propriedades de segurança. Entretanto, por razões metodológicas, considera-se como propriedades de animação com limites temporais ou propriedades de temporalidade.

4.4.1.4 Propriedades de Ausência de Deadlock

A ausência de *deadlock* é uma propriedade especial que descreve que o sistema nunca se encontrará em um estado em que ele esteja impedido de progredir. A ausência de *deadlock* é uma propriedade muito importante, principalmente para sistemas que devem executar indefinidamente, como é o caso dos sistemas de controle e os protocolos.

A ausência de *deadlock* poderia ser vista como uma propriedade de segurança, na medida que descreve que algo ruim nunca irá acontecer. Entretanto, deve-se observar que não é possível se expressar a ausência de *deadlock* através de uma fórmula do tipo $\mathtt{A}\Box\phi$, com ϕ expressando uma fórmula de estado.

De modo similar, esta propriedade pode ser vista como uma propriedade de animação, pois a ausência de *deadlock* representa que em qualquer estado é sempre verdade que existe um outro estado que pode ser futuramente alcançado. Mais uma vez, se ϕ é uma fórmula de estado, não é possível expressar a ausência de *deadlock* através de uma fórmula do tipo $\mathtt{A}\Diamond\phi$.

De fato, a ausência de *deadlock* possui características de propriedades de segurança e de animação, embora não possa ser completamente enquadrada como qualquer delas. Em CTL a ausência de *deadlock* é representada através da fórmula $A\Box E\Diamond true$, que descreve que para todos os caminhos é sempre verdade que o sistema pode avançar.

Como o aninhamento de operadores de caminho não é possível em $\mathscr{L}_s$, a lógica incluiu o predicado $deadlock$ para representar o estado em que não existe nenhum estado sucessor válido, ou seja, que o sistema não mais pode avançar. Assim, $\mathscr{L}_s$ representa a ausência de deadlock através da fórmula $A\Box !deadlock$

4.4.1.5 Propriedades de Justiça

Uma propriedade de justiça descreve que, sob certas circunstâncias, um evento irá ocorrer (ou não irá ocorrer) ciclicamente (infinitamente futuramente), ou seja, propriedades de justiça são usadas para representar que se algo pode acontecer infinitas vezes então ocorrerá infinitas vezes. Devido a esta característica, uma propriedade de justiça é também denominada propriedade de alcance repetitivo (*repeated reachability*). A linguagem $\mathscr{L}_s$, como qualquer subconjunto de CTL, não possui expressividade suficiente para descrever esta categoria de propriedades. Entretanto, é possível a verificação desta classe de propriedades através do UPPAAL, de modo indireto, utilizando-se de um método conhecido por autômato observador ou de teste. Neste caso, o autômato de teste é usado para, através de sincronizações, registrar o caminho seguido pelo autômato que se deseja investigar, incluindo a existência de ciclos que definam a ocorrência de uma propriedade de justiça (BFB01).

4.4.2 Verificação do Exemplo da Bomba

Para o exemplo do conjunto bomba temporizada e sensor, um conjunto de propriedades foram verificadas para garantir a correção do sistema. Para realizar estas verificações, modificamos o sistema através da adição de um relógio global (`g_time`) e um relógio local ao modelo da bomba (`c_time`). Estes relógios foram adicionados apenas com o propósito de se descrever propriedades que levem em conta os seus valores, quando um determinado estado do relógio for alcançado. O relógio global é reiniciado quando o sensor termina a leitura e processamento da variável controlada (transição entre os estados `Read` e `Start`). O relógio `c_time` marca o tempo decorrido desde o momento em que a bomba tenha sido desligada e é reiniciado durante a transição entre os estados `On` e `Off`.

As propriedades desejadas para o sistema estão listadas à seguir.

Propriedades de Segurança

Propriedade 1 (Tempo Máximo de Processamento de Sinal)**.** *O tempo de processamento e leitura do sinal é sempre menor ou igual a 5 unidades de tempo.*

Esta propriedade pode ser representada pela fórmula `A[](sensor.Read imply sensor.c_sensor <= 5)`. O tempo máximo de processamento é uma invariante do sistema bomba/sensor.

Propriedades de Alcançabilidade

Propriedade 2 (Acionamento Contínuo)**.** *É possível que a bomba permaneça sempre ligada, ou seja, que a bomba seja ligada no instante imediatamente posterior àquele em que ela tenha sido desligada.*

Representamos a propriedade pela fórmula `E<>(pump.On && pump.c_time == 0)`, que denota que existe uma execução, na qual a bomba será ligada (estado `On`) em 0 unidades de tempo após ela ter sido desligada (`pump.c_time` $==$ `0`).

Propriedades de Ausência de Deadlock

Propriedade 3 (Ausência de Deadlock)**.** *O conjunto bomba sensor é livre de deadlock.*

Para representar esta propriedade usamos a seguinte fórmula em $\mathscr{L}_s$: `A[]!deadlock`.

Propriedades de Animação

Propriedade 4 (Tempo Máximo entre Acionamentos)**.** *Uma vez que a bomba se encontra desligada, ela estará ligada em no máximo 15 unidades de tempo.*

A fórmula `(pump.Off && pump.c_time == 0) --> (pump.On && pump.c_time <= 15)` representa esta propriedade. Dado que a bomba está no nó `Off` e o relógio `c_time` em zero, é sempre verdade que futuramente a bomba estará no nó `On`, e o relógio `c_time` com, no máximo, valor 15.

Propriedade 5 (Acionamento Imediato)**.** *Caso o sensor dispare a bomba será imediatamente ligada.*

Representamos a propriedade através da fórmula `sensor.Start --> pump.On && g_time == 0`. O sensor no estado de partida, (estado `Start`) sempre leva a bomba ao estado ligado após 0 unidades de tempo, situação representada pelo relógio `g_time` com valor zero.

As propriedades 1, 2, 4 e 5 foram verificadas. Entretanto, ao submeter a propriedade 3 o verificador mostra que o modelo falha em satisfazer a propriedade, pois quando a bomba já se encontra ligada, ela não consegue receber mensagens de acionamento do sensor. Como contra exemplo, o verificador apresentou a seguinte execução:

$$\begin{aligned}\xi = \langle &(\{\texttt{Off},\texttt{Idle}\},\{\texttt{c\_pump}=0.0,\texttt{c\_time}=0.0,\texttt{c\_sensor}=0.0,\texttt{g\_time}=0.0\}),\\ &(\{\texttt{Off},\texttt{Read}\},\{\texttt{c\_pump}=3.0,\texttt{c\_time}=3.0,\texttt{c\_sensor}=0.0,\texttt{g\_time}=3.0\}),\\ &(\{\texttt{Off},\texttt{Start}\},\{\texttt{c\_pump}=0.0,\texttt{c\_time}=3.0,\texttt{c\_sensor}=0.0,\texttt{g\_time}=3.0\}),\\ &(\{\texttt{On},\texttt{Idle}\},\{\texttt{c\_pump}=0.0,\texttt{c\_time}=3.0,\texttt{c\_sensor}=0.0,\texttt{g\_time}=3.0\}),\\ &(\{\texttt{On},\texttt{Read}\},\{\texttt{c\_pump}=3.0,\texttt{c\_time}=6.0,\texttt{c\_sensor}=0.0,\texttt{g\_time}=6.0\}),\\ &(\{\texttt{On},\texttt{Start}\},\{\texttt{c\_pump}=3.0,\texttt{c\_time}=6.0,\texttt{c\_sensor}=0.0,\texttt{g\_time}=6.0\}),\\ &\texttt{deadlock}\ \rangle\end{aligned}$$

O modelo corrigido apresentado na figura 4.5 satisfaz todas as propriedades apresentadas.

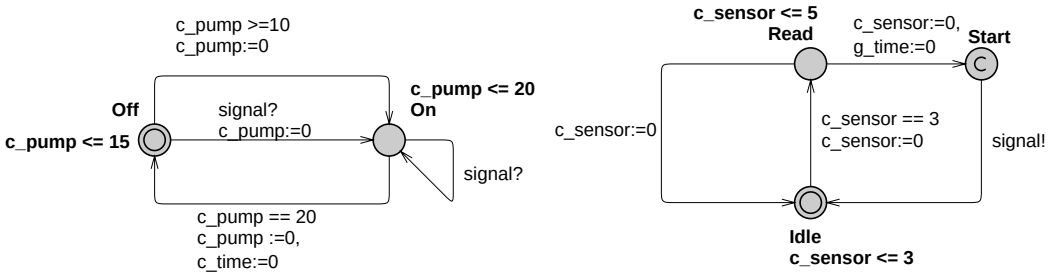

Figura 4.5 Modelo de uma Bomba Temporizada com Sensor

CAPÍTULO 5

Especificação Formal da Camada de Acesso ao Meio do IEEE 802.11

O mundo é a minha representação.

—ARTHUR SCHOPENHAUER (Filósofo Alemão)

Não há fatos, só interpretações.

—FRIEDRICH NIETZSCHE (Filósofo Alemão)

Este capítulo descreve como foi realizada a especificação da função de controle de acesso ao meio do padrão 802.11. A subcamada MAC foi modelada através de seis autômatos. Três deles representam as tarefas que descrevem o comportamento de estações que operam durante o CP (*Contention Period*), conforme as regras da DCF (seção 5.2). Os demais representam, respectivamente, o comportamento das estações durante o CFP (*Contention Free Period*) sob as regras da PCF (seção 5.4), o comportamento da estação que atua como *Point Coordinator* (seção 5.3), e o comportamento do meio físico (seção 5.6). Na próxima seção (seção 5.1) são apresentadas as variáveis e constantes compartilhadas, que foram utilizadas para a especificação.

5.1 Constantes e Variáveis Globais

Para a especificação foi utilizada uma série de constantes e variáveis declaradas globalmente, e, portanto, acessíveis por todos os autômatos.

O primeiro grupo de variáveis e constantes dizem respeito às características do meio físico. Estas constantes são apresentadas na tabela 5.1.

O segundo grupo de constantes descreve o tamanho e o tempo de transmissão dos quadros que participam da função de controle de acesso ao meio da especificação IEEE 802.11. Estas constantes são apresentadas nas tabelas 5.2 e 5.3.

Tabela 5.1 Parâmetros característicos de cada camada física

Constante	**Descrição**	**Valor**
`aSlotTime` `aSIFSTime`	Constantes dependentes da camada física. Consideram o tempo de propagação do sinal no meio e o tempo de processamento do sinal na subcamada PHY	50 28 (Valores para FHSS — Frequency-Hopping Spread Spectrum — PHY)
`SIFS`	*Short Interfame Space*	`aSIFSTime`
`PIFS`	*PCF Interfame Space*	`aSIFSTime` + `aSlotTime`
`DIFS`	*DCF Interfame Space*	`aSIFSTime` +2× `aSlotTime`
`dataRate`	Taxa de transmissão (bits por μs)	1

Tabela 5.2 Tamanho dos Quadros (em bits)

Constante	**Descrição**	**Valor**
`ACK_SIZE`	Tamanho do quadro de controle ACK	112
`CTS_SIZE`	Tamanho do quadro de controle CTS	112
`RTS_SIZE`	Tamanho do quadro de controle RTS	160
`CF_END_SIZE`	Tamanho do quadro de controle CF-End	160
`CF_END$CF_ACK_SIZE`	Tamanho do quadro de controle CF-End+CF-Ack	160
`DATA_MIN_SIZE`	Tamanho do menor quadro de dados	272
`DATA_MAX_SIZE`	Tamanho do maior quadro de dados	18768

Tabela 5.3 Tempo de Transmissão dos Quadros (em μs)

Constante	**Descrição**	**Valor**
ACK_TIME	Tempo necessário para transmissão de quadro de controle ACK	ACK_SIZE/ dataRate
BEACON_TIME	Tempo necessário para transmissão do quadro de gerenciamento BEACON	BEACON_SIZE/ dataRate
CTS_TIME	Tempo necessário para transmissão de quadro de controle CTS	CTS_SIZE/ dataRate
RTS_TIME	Tempo necessário para transmissão de quadro de controle RTS	RTS_SIZE/ dataRate
CF_END_TIME	Tempo necessário para transmissão de quadro de controle CF-End	CF_END_SIZE/ dataRate
CF_END$CF_ACK_TIME	Tempo necessário para transmissão de quadro de controle CF-End+CF-Ack	CF_END$CF_ACK_SIZE/ dataRate
MPDU_MIN_TIME	Tempo necessário para transmissão do menor quadro de dados	DATA_MIN_SIZE/ dataRate
MPDU_MAX_TIME	Tempo necessário para transmissão do maior quadro de dados	DATA_MAX_SIZE/ dataRate
MIN_DATA_TIME	Tempo necessário para transmissão dos quadro especiais de dados sem nenhuma informação (Null Function, CF-Ack, CF-Poll e CF-Ack+CF-Poll).	DATA_MIN_SIZE/ dataRate

O terceiro e quarto grupo de variáveis (tabela 5.4 e 5.5) são formados por variáveis canais (`chan`), que modelam a interação dos autômatos que representam as estações, com o autômato que representa o meio físico e a interação do meio físico com as estações. O uso destas variáveis canais permitiu, além da modelagem e verificação de situações de colisão, a modelagem da função CCA da camada física. Canais urgentes foram usados para modelar o repasse das mensagens lançados no meio para as estações destinos. O uso de canais urgentes teve como motivo o fato de, uma vez que a mensagem esteja disponível no meio físico, a estação para a qual esta mensagem destina-se, deve consumi-la de imediato. Deste modo, esta espécie de canal torna-se como a alternativa indicada para a situação a ser modelada.

Finalmente, tem-se um grupo de variáveis auxiliares utilizadas para diversos fins, tais como o registro do modo de operação do protocolo, descritas na tabela 5.6.

5.2 Modelo das Estações sob a DCF

Nesta seção, apresentamos os três autômatos com tempo que são utilizados para representar o comportamento das estações durante o período de contenção. Neste período, as estações atuam de acordo com as regras da DCF. A subseção 5.2.1 apresenta o modelo que representa a tarefa de recepção de mensagens em cada uma das estações. Em seguida, na subseção 5.2.2, encontra-se a descrição do autômato que modela a tarefa de envio de mensagens. Finalmente, a subseção 5.2.3 descreve a tarefa de escuta do meio realizada pelas estações, de acordo com a política de sensoreamento físico e virtual definida para o protocolo.

5.2.1 Modelo de Recepção de Mensagens de Estações sob a DCF

Na figura 5.1 pode-se observar o autômato que apresenta o comportamento da tarefa de recepção de mensagens no modo DCF.

Para a modelagem desta tarefa, foi declarada localmente uma variável relógio de nome `local`. Esta variável é utilizada para controlar o comportamento temporal durante o processo de comunicação. Deste modo, a variável `local` monitora a duração do espaçamento entre quadros e o tempo de transmissão dos quadros ACK e CTS.

A recepção de mensagens é especificada através dos nós `Init`, `ReceivingRTS`, `WaitSIFS`, `SendingCTS`, `WaitMPDU`, `ReceivingMPDU`, `ReceivedMPDU`, `MPDUOk` e `SendingACK`.

O nó `Init` é o nó inicial do autômato e representa o momento em que a tarefa de recepção

Tabela 5.4 Canais que modelam a iteração das estações com o meio físico

Constante	**Descrição**
`iniACKm` e `endACKm`	Modelam o início e o término do envio de quadros ACK de uma estação para o meio físico.
`iniCTSm` e `endCTSm`	Modelam o início e o término do envio de quadros CTS de uma estação para o meio físico.
`iniMPDUm` e `endMPDUm`	Modelam o início e o término do envio de quadros de dados (MPDU) da estação para o meio físico.
`iniRTSm` e `endRTSm`	Modelam o início e o término do envio de quadros RTS de uma estação para o meio físico.
`iniBeacon` e `endBeacon`	Modelam o início e o término do envio do quadro Beacon do PC para o meio físico.
`iniCF_End` e `endCF_End`	Modelam o início e o término do envio do quadro CF-END do PC para o meio físico.
`iniCF_End$CF_Ack` e `endCF_End$CF_Ack`	Modelam o início e o término do envio do quadro CF-ACK+CF-END do PC para o meio físico.
`iniCF_Pollm`	Modela o início do envio do quadro CF-Pool do PC para o meio físico.
`iniData$CF_Pollm`	Modela o início do envio do quadro combinado Data+CF-Poll do PC para o meio físico.
`iniCF_Ack$CF_Pollm`	Modela o início do envio do quadro combinado Cf-Ack+CF-Poll do PC para o meio físico.
`iniData$CF_Ack$CF_Pollm`	Modela o início do envio do quadro combinado Data+CF-Ack+CF-Poll do PCF para o meio físico.
`iniNullm`	Modela o início do envio do quadro Null de uma estação que opera sob a PCF para o meio físico.
`iniData$CF_Ackm`	Modela o início do envio do quadro combinado Data+CF-Ack de uma que opera sob a PCF para o meio físico.
`iniCF_Ackm`	Modela o início do envio do quadro combinado CF-Ack de uma estação que opera sob a PCF para o meio físico

Tabela 5.5 Canais que modelam a iteração das estações com o meio físico

Constante	Descrição
`iniACK` e `endACK`	Modelam o início e o término do envio de quadros ACK do meio físico para a estação.
`iniCTS` e `endCTS`	Modelam o início e o término do envio de quadros CTS do meio físico para a estação.
`iniMPDU` e `endMPDU`	Modelam o início e o término do envio de quadros de dados (MPDU) do meio físico para a estação.
`iniRTS` e `endRTS`	Modelam o início e o término do envio de quadros RTS do meio físico para a estação.
`iniCF_Poll`	Modelam o início do repasse de um quadro CF-Poll do meio físico para uma estação.
`iniData$CF_Poll`	Modelam o início do repasse de um quadro Data+CF-Poll do meio físico para uma estação.
`iniCF_Ack$CF_Poll`	Modelam o início do repasse de um quadro CF-ACK+CF-Poll do meio físico para uma estação.
`iniData$CF_Ack$CF_Poll`	Modelam o início do repasse de um quadro Data+CF-ACK+CF-Poll do meio físico para uma estação.
`iniData$CF_Ack`	Modelam o início do repasse de um quadro Data+CF-ACK do meio físico para o PC.
`iniNull`	Modelam o início do repasse de um quadro Null do meio físico para o PC.
`iniCF_Ack`	Modelam o início do repasse de um quadro CF-Ack do meio físico para as estações.

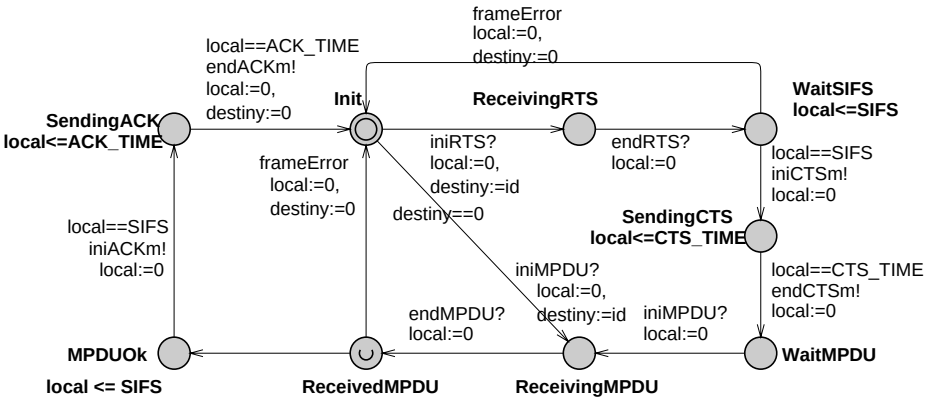

Figura 5.1 Tarefa de recepção de mensagens das estações na DCF

Tabela 5.6 Demais variáveis e constantes globais

Constante	**Descrição**
`cfAck`	Canal utilizado para sinalizar que uma mensagem de confirmação foi enviada em quadro combinado
`noCfAck`	Canal utilizado para informar que na mensagem enviada não foi incluida nenhum quadro de confirmação.
`beacon`	Canal utilizado para informar a todas as estações que o PC enviou um quadro *beacon*
`mode` `PCF` e `DCF`	Variável e constantes lógicas utilizadas para registrar o modo atual de operação do protocolo
`MAX_STA` e `nav[MAX_STA]` `BUSY` e `IDLE`	Arranjo de variáveis lógicas, uma para cada estação, que armazena a visão local do estado do meio físico, representado pelas constantes `BUSY` e `IDLE`
`idle[MAX_STA]` `busy[MAX_STA]`	Canais usados para comunicação entre o autômato que modela a função de sensoreamento e as outras tarefas da estação
`phyIdle` e `phyBusy` `navIdle`e `navBusy`	Canais utilizados para sinalizar às funções de sensoriamento da mudança do estado do meio
`CTSTimeout` `ACKTimeout`	Constantes que modelam o *timeout* da espera pelos quadros CTS e ACK
`CP_MIN_DURATION` `CFPMaxDuration` `CFPRate`	Constantes que controlam o intervalo de alternância entre as funções de coordenação do uso do meio físico
`destiny`	Variável utilizada para que uma tarefa de leitura indique qual estação está recebendo a mensagem
`frameError`	Variável booleana que indica a possibilidade de chegada de quadros corrompidos

de mensagens está apenas escutando o meio, aguardando pela chegada de alguma mensagem. O nó `ReceivingRTS` representa o período durante o qual a estação está recebendo um quadro do tipo RTS. Por sua vez, o nó `WaitSIFS` representa o período em que a estação está aguardando o término do período de inatividade do meio para responder à estação transmissora. Em `SendingCTS`, apresenta-se o intervalo durante o qual a estação receptora está enviando um quadro do tipo CTS em resposta a um quadro RTS previamente recebido. No nó `WaitMPDU` modela-se a espera da estação receptora pela chegada de uma mensagem de dados (MPDU). `ReceivingMPDU` é o nó que modela o período durante o qual a estação receptora está recebendo a mensagem de dados propriamente dita. Em `ReceivedMPDU`, tem-se o nó em que a estação verifica a consistência da mensagem recebida. O nó `MPDUOk` representa a ocorrência de uma transmissão com sucesso, ou seja, que a mensagem foi recebida sem erros. Finalmente, o nó `SendingACK` descreve o período de envio do quadro de confirmação ACK pela estação receptora, indicando ao transmissor que a transmissão foi realizada com sucesso.

O comportamento deste autômato pode ser assim descrito: o autômato inicia no nó `Init` e aí permanece até que a estação detecte o início da transmissão de um quadro RTS (ativação da aresta com ação de sincronização `iniRTS?`) ou de um quadro de dados (MPDU) (ativação da aresta com ação de sincronização `iniMPDU?`). No primeiro caso, o autômato passa a consumir o quadro RTS (permanência no nó `ReceivingRTS`) e ao fim da leitura deste quadro irá para o nó `WaitSIFS` (ativação da aresta com a sincronização `endRTS?`). O autômato permanecerá neste nó por no máximo um SIFS, condição garantida pelo invariante `local<=SIFS` associado a este nó.

Em seguida, a estação irá optar de modo não determinista por retornar ao nó inicial, representando o fato do quadro de controle RTS não ter sido recebido com sucesso, ou por responder ao envio do RTS com um quadro CTS (indicando o fato que a requisição de reserva de uso do meio foi transmitida com sucesso). Nesta última situação, o autômato irá transitar para o nó `SendingCTS` exatamente após a passagem de um SIFS, condição garantida pela combinação do guarda `local==SIFS` associado à aresta e ao invariante `local<=SIFS` associado ao nó origem. Nesta aresta, o autômato irá sincronizar com o meio através do canal `iniCTSm!`, representando o fato de que a tarefa de recepção inicia a transmissão de um CTS. O autômato permanece no nó `SendingCTS` o tempo necessário para a transmissão de um quadro de controle CTS. Em seguida, o autômato que modela a recepção de mensagens sinaliza ao autômato que modela o meio, o término do envio do CTS (aresta com ação de sincronização `endCTSm!`) e passa a esperar pela chegada do quadro de dados (MPDU). O autômato permanece no nó `WaitMPDU` até que perceba que um MPDU está sendo enviado. Esta percepção é modelada pela ativação da aresta com ação de sincronização `iniMPDU?`, que leva o autômato ao nó

ReceivingMPDU. Deve-se observar que este nó é alcançável a partir do nó inicial através de uma aresta com a mesma ação de sincronização (iniMPDU?), modelando assim os modos de operação DCF Básico e DCF com RTS/CTS. O autômato permanece no nó ReceivingMPDU até a detecção do fim da transmissão do quadro, representado pela transição com sincronização no canal endMPDU?, que leva o autômato para o nó ReceivedMPDU. Neste nó, o autômato pode escolher de modo não determinista por transitar para o nó inicial, representando a situação em que a mensagem recebida esteja corrompida, ou para o nó MPDUOk, representanto a situação em que a recepção do MPDU teve sucesso. Neste último caso, a estação irá aguardar por exatamente um SIFS, conforme descrito pela invariante do nó MPDUOk (local<=SIFS) e pelo guarda da aresta que parte deste nó (local==SIFS). Esta transição modela ainda o início do envio do quadro ACK, através da ação de sincronização iniACKm!. O autômato permanece no nó SendingACK o tempo necessário para o envio do quadro de aceitação (ACK) quando, finalmente, retorna para o nó inicial (aresta com ação de sincronização endACKm!).

5.2.2 Modelo de Envio de Mensagens de Estações sob a DCF

Para melhor compreensão do autômato que modela a tarefa de envio de mensagem, este será dividido em dois grupos. O primeiro deles é formado pelos nós InitBackoff, WaitIdle, WaitIFS1, ChooseBckOffValue, Backoff, Freeze, WaitIFS2 e EndBackoff e representa o procedimento de *backoff*. O segundo grupo é formado pelos demais nós e modela o processo efetivo de envio de mensagem. A figura 5.2 apresenta o autômato completo.

As próximas subseções discutem o comportamento de cada um dos grupos de nós que compõem o autômato.

5.2.2.1 Procedimento de *backoff*

O procedimento de *backoff* pode ser iniciado através de quatro arestas. A primeira delas parte do nó InitDCF e representa a situação em que uma estação que deseja transmitir encontra o meio ocupado (aresta com guarda nav==BUSY). A segunda aresta parte do nó WaitDIFS e será ativada caso a estação que espera pela passagem do período de inatividade do meio (DIFS) perceba que, durante este período, uma outra estação iniciou uma transmissão (ativação da aresta com ação de sincronização busy?). Uma terceira aresta leva do nó WaitCTS para o nó NoCTS e deste para o início do *backoff*. Esta aresta modela a situação em que a estação origem, após enviar um quadro RTS, não obtém um CTS como resposta. Finalmente, a quarta aresta parte do nó NoACK e representa o fato de uma estação não receber um quadro de confirmação, após a transmissão de MPDU, indicando uma falha no processo de comunicação.

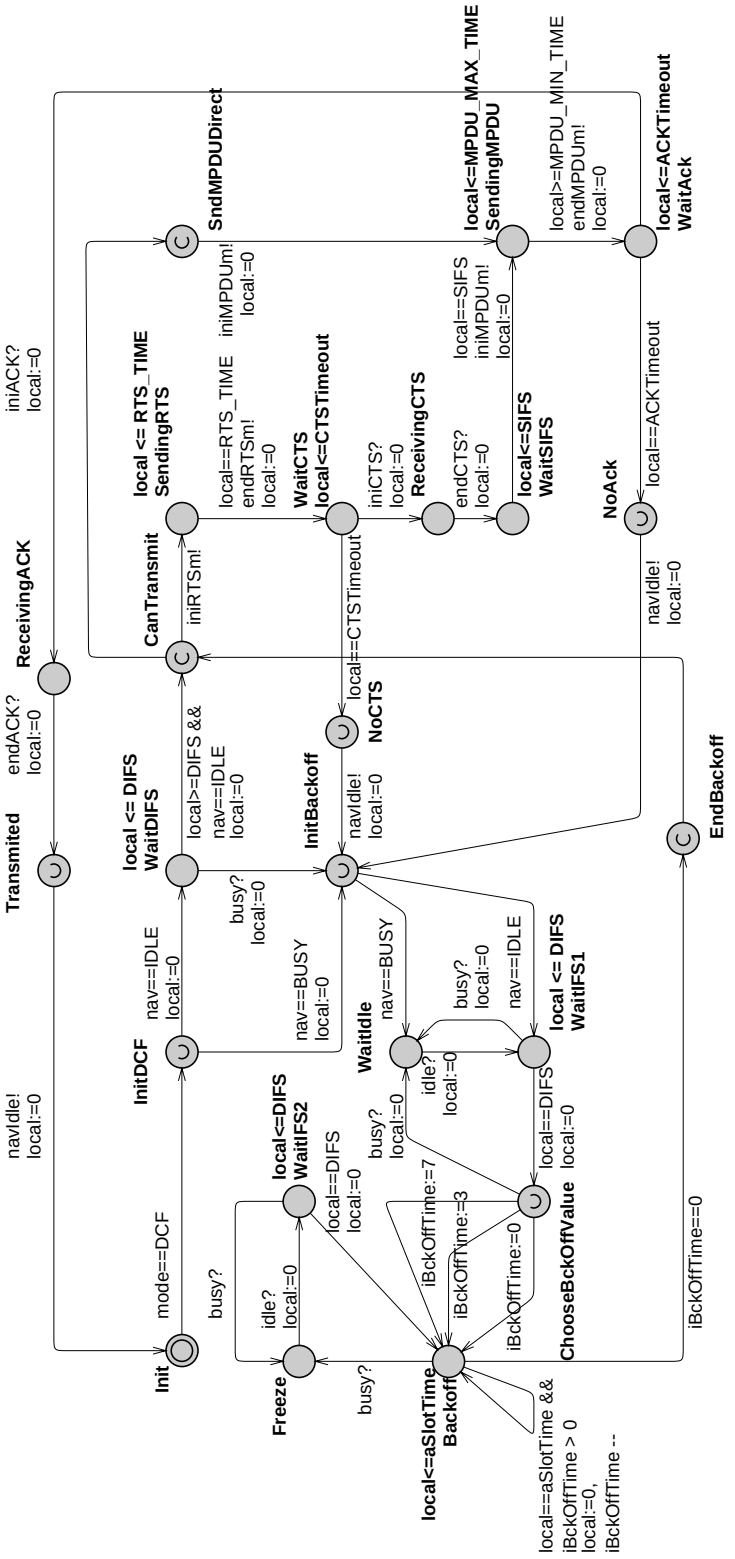

Figura 5.2 Tarefa de envio de mensagens das estações na DCF

Uma vez no procedimento de *backoff*, a estação verifica de imediato (nó urgente) se o meio se encontra ocupado. Caso o meio se encontre ocupado, o autômato irá transitar para o nó `WaitIDLE` (aresta com guarda `nav==BUSY`), onde passará a aguardar até que o meio volte a ficar inativo. Uma vez que o meio se torne livre (ativação da aresta com ação de sincronização `idle?`), o autômato migra para o nó `WaitIFS1`. Este nó representa o período em que a estação aguarda a passagem de um DIFS de inatividade, antes de iniciar o período de espera aleatória do *backoff*. O nó `WaitIFS1` pode ser alcançado diretamente a partir de `InitBackoff` caso o meio se encontre livre quando o procedimento de *backoff* for iniciado (aresta com guarda `nav==IDLE`).

Se durante o período em que a tarefa aguarda a passagem do DIFS, o meio voltar a ficar ocupado (aresta com ação de sincronização `idle?`) o autômato irá transitar para o nó `WaitIDLE`, onde reiniciará a espera pela inatividade do meio. Entretanto, caso o meio permaneça livre por todo o período, o autômato migrará para o nó `ChooseBckOffValue`, conforme definido pelo invariante `local<=DIFS` associado ao nó `WaitIFS1` e pelo guarda `local==DIFS` da aresta que une os nós `WaitIFS1` e `ChooseBckOffValue`.

O nó `ChooseBckOffValue` é um nó urgente que possui várias transições para o nó `Backoff`. Estas diversas transições modelam a escolha de um tempo aleatório de espera, a ser armazenado na variável relógio local `iBckOffTime` que modela o relógio de *backoff*. A permanência no nó `Backoff` é de no máximo um *aSlotTime*, como determina a invariante `local<=aSlotTime`. Caso o meio permaneça livre durante todo um *aSlotTime*, a aresta que parte deste nó e a ele retorna é habilitada (guarda `local<=aSlotTime`). Deste modo, o relógio de *backoff* é decrementado (ação `iBckOffTime--`). Entretanto, durante a espera pela passagem de um *aSlotTime* no nó `backoff`, uma estação pode perceber que o meio voltou a ficar ocupado, devendo congelar o seu relógio de *backoff*. Neste caso, o autômato irá transitar para o nó `Freeze` (aresta com ação de sincronização `busy?`), e a variável `iBckOffTime` deixará de ser decrementada a cada *aSlotTime*.

O nó `Freeze` representa o fato do relógio de *backoff* estar congelado, não sofrendo assim qualquer variação em seu valor. O autômato permanecerá em `Freeze` até que o meio volte a ficar livre, quando migrará para o nó `WaitIFS2`. O autômato deixa este nó apenas quando o meio volte a ficar ocupado, caso em que o autômato irá retornar ao nó de relógio de *backoff* congelado (nó `Freeze` através da aresta com sincronização `busy?`), ou após a passagem de um período de inatividade DIFS (representado pelo invariante do nó `local<=DIFS` e guarda `local==DIFS`), caso em que o autômato retorna para o nó `Backoff`. Neste nó, ele volta a aguardar por períodos *aSlotTime* de inatividade para decrescer o relógio de *backoff*.

A terceira aresta de saída do nó *backoff* somente é habilitada quando o relógio de *backoff*

chega ao valor zero (guarda `iBckOffTime==0`). Neste caso, o período de *backoff* chegou ao fim (nó `EndBackOff`) e a estação começa imediatamente a transmitir. Esta necessidade de transmissão imediata foi representada por um nó *commited* no final do *backoff*. Ao sair do nó do procedimento de *backoff*, o autômato migrará para o nó onde inicia a utilização do meio (`CanTransmit`).

5.2.2.2 Processo de transmissão de mensagem

O segundo grupo de nós representa a transmissão efetiva da mensagem. Este grupo é composto pelos nós:

- `InitDCF` - representa o fato da tarefa de transmissão de mensagem possuir pacotes a trasmitir;
- `WaitDIFS` - modela o nó em que a tarefa de transmissão de mensagem está aguardando a passagem de um período de inatividade do meio (DIFS);
- `CanTransmit` - modela o instante exato em que se dá o início da transmissão de um quadro;
- `SendingRTS` - representa o período durante o qual a tarefa de transmissão está enviando um quadro *Request To Send* para alocação do meio.
- `WaitCTS` - apresenta o período durante o qual a estação aguarda pelo envio de um quadro *Clear To Send*;
- `ReceivingCTS` - modela o período durante o qual a tarefa está recebendo um quadro CTS, que indica que a reserva do meio foi realizada com sucesso;
- `WaitSIFS` - representa a janela temporal de inatividade entre a seqüência atômica de comunicação, formada pela recepção do CTS, e o envio do quadro de dados;
- `SendingMPDUDirect` - indica que a tarefa de envio de mensagens optou pelo DCF Básico. Não há consumo de tempo neste estado, o que é representado pelo uso de um nó *commited*;
- `SendingMPDU` - modela a janela temporal durante a qual os dados estão efetivamente sendo enviados;
- `WaitAck` - modela a espera da tarefa de envio de mensagens por um quadro de confirmação (ACK);

- `ReceivingAck` - representa a recepção do quadro ACK;
- `Transmitted` - indica uma transmissão realizada com sucesso;
- `NoCTS` - modela a não recepção de um quadro CTS após a tarefa ter enviado um RTS correspondente. Neste caso, a tarefa deverá entrar de imediato em *backoff* (nó *urgent*);
- `NoAck` - indica que a transmissão de um quadro de dados não foi seguida pelo ACK correspondente. Nesta situação, a tarefa deverá entrar de imediato em *backoff* (nó *urgent*).

Para enviar uma mensagem, a tarefa de envio deverá verificar se o meio está livre (guarda `nav == IDLE` da aresta entre os nós `InitDCF` e `WaitDIFS`), aguardar o período de inatividade do meio (permanência no nó `WaitDIFS`) e, caso o meio permaneça livre durante todo o período, começar de imediato a transmissão de um quadro (migração para o nó `CanTransmit`).

O nó `CanTransmit` é um nó *commited* e indica que esta é uma locação onde não são permitidas intercalações nem transições retardo. Isto modela o fato do início da transmissão acontecer imediatamente após a passagem do período de inatividade do meio, ou após o término do *backoff* (caso este nó seja alcançado através da aresta que parte de `EndBackoff`). Uma vez em `CanTransmit`, o autômato poderá optar por realizar uma transmissão no modelo DCF Básico, caso em que irá migrar para o nó `SndMPDUDirect`, ou por realizar uma transmissão DCF com CTS/RTS, situação modelada por uma transição para o nó `SendingRTS`.

A migração para o nó `SendingRTS` se dá através de uma transição em que acontece uma operação de escrita sobre o canal `iniRTSm`. Esta operação indica o início da trasmissão de um quadro RTS com o canal `iniRTSm` sendo utilizado, portanto, para realizar a sincronização entre a tarefa de envio de mensagem e o meio físico. O autômato irá permanecer no nó `SendingRTS` pelo tempo necessário à transmissão de um RTS (modelado pela invariante `local <= RTS_TIME` e pelo guarda `local == RTS_TIME`) e migrará para o nó `WaitCTS`. Esta migração acontece através de uma transição ação que indica o término do envio do quadro RTS (ação de sincronização `endRTSm!` associada à aresta). O autômato irá permanecer por no máximo `CTSTimeout` no nó `WaitCTS` e deixará este nó devido a passagem de `CTSTimeout` (caso em que o autômato migra para o nó `NoCTS` e inicia um procedimento de *backoff*) ou devido a chegada de um quadro CTS. Neste último caso, a saída do nó se dará através de uma ação de sincronização com o meio físico, representada pela operação de leitura sobre o canal `iniCTS?`. O autômato permanece no nó `ReceivingCTS` até que nova sincronização através do canal `endCTS?` indique o término da transmissão do quadro CTS, e então vai para o nó `WaitSIFS`.

Após a espera de um espaçamento entre quadros do tipo SIFS no nó `WaitSIFS` (novamente caracterizando uma seqüência atômica de troca de quadros), a estação pode finalmente proceder com o envio dos dados em si. Isto é representado pela transição com ação de sincronização `iniMPDUm!` que conecta o nó `WaitSIFS` ao nó `SendingMPDU`. Deve-se notar que uma transição similar conecta o nó `CanTransmit` ao nó `SendingMPDU` (via nó *commited* `SndMPDUDirect`), modelando o envio de dados sem a reserva de uso do meio através de RTS/CTS. O período de permanência no nó `SendingMPDU` pode variar entre `MPDU_MIN_TIME` e `MPDU_MAX_TIME`, indicando que o tempo de envio do MPDU irá depender do tamanho do quadro que está sendo transmitido (modelado pela invariante `local<=MPDU_MAX_TIME` e guarda `local>=MPDU_MIN_TIME`).

Em seguida, o autômato migra para o nó `WaitAck` modelando o fim da transmissão. Isto é representado pela sincronização entre a tarefa de envio e o meio físico através de uma operação de escrita sobre o canal `endMPDUm`. O autômato permanece em `WaitAck` até que se inicie a recepção de um quadro de confirmação (transição para o nó `ReceivingACK`, com ação de sincronização `iniAck?`) ou que aconteça um *timeout* (`ACKTimeout`). Este último caso representa uma falha na transmissão do MPDU e o autômato deve migrar para o nó `NoAck` (nó *urgent*), quando então um procedimento de *backoff* será imediatamente disparado.

Caso a transmissão do MPDU tenha se dado com sucesso, a tarefa receberá o quadro ACK até o seu término, representado através de uma leitura sobre o canal `endACK?`. Neste momento, o autômato migrará para o nó `Transmited`, que indica que a transmissão ocorreu com sucesso e daí, sem que se passe tempo, para o nó inicial do autômato.

5.2.3 Modelo de Sensoreamento do Meio

Para a representação da função de sensoreamento do meio foi definido um autômato que se comunica com as tarefas de envio de mensagens através de dois canais de comunicação e de uma variável lógica. Os canais de comunicação são utilizados para informar às tarefas de envio de mensagens mudanças no estado do meio físico. A variável armazena o estado do meio (`BUSY` ou `IDLE`) conforme a percepção do mecanismo de sensoreamento do meio físico. Os canais e a variável lógica são informados como parâmetros no momento da instanciação das funções de sensoreamento do meio (`bool csm; urgent broadcast chan busy; urgent broadcast chan idle`), de forma a definir qual tarefa de envio está relacionada a qual tarefa de sensoreamento. Isto modela o fato de que cada estação possui a sua própria função de sensoreamento e, portanto, uma visão independente do estado do meio físico. Assim, são instanciadas tantas tarefas de sensoreamento do meio quantas forem as tarefas de envio de

mensagens.

A função de sensoreamento do meio é representada pelo autômato da figura 5.3 e congrega o sensoreamento físico e o sensoreamento virtual. Este autômato é composto pelos seguintes nós:

- `Idle` - nó inicial do autômato, que representa o estado em que a função de sensoreamento percebe que o meio está livre;
- `BusingNAV` - nó que exibe o instante em que a função de sensoreamento virtual percebe que o meio se tornou ocupado;
- `Nav` - representa o estado em que a função de sensoreamento virtual registra o estado do meio como ocupado, tendo feito a notificação do fato às tarefas de envio de mensagens;
- `BusingPHY` - modela o momento em que a função de sensoreamento físico percebe a utilização do meio;
- `Phy` - este nó representa o registro do estado do meio como ocupado pela função de sensoreamento.
- `IdleingNAV` - nó que representa o instante em que a função de sensoreamento virtual percebe que o meio se tornou livre;
- `IdleingPhy` - modela o momento em que a função de sensoreamento física percebe que o meio se tornou ocioso;
- `Both` - estado em que as duas funções de sensoreamento registram a situação do meio como ocupado.

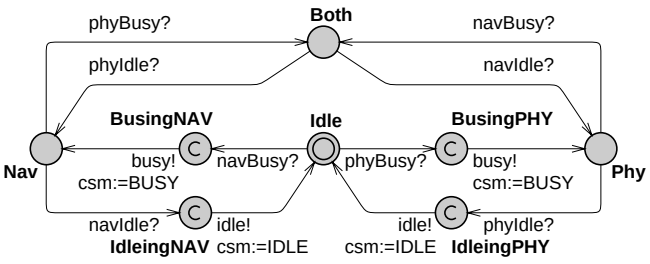

Figura 5.3 Função de Sensoreamento do Meio

Os nós `BusingNAV`, `BusingPHY`, `IdleingNAV` e `IdleingPHY` conectam ações de identificação de mudança do estado do meio (ações de leitura sobre os canais `navBusy`, `navIdle`, `phyBusy` e `phyIdle`) com a notificação instantânea desta percepção para as funções de envio de mensagens (escrita sobre os canais `busy` e `nav`). Assim, como forma de modelar esta comunicação instantânea, fêz-se uso de nós *commited*.

Este autômato inicia no nó `Idle`, indicando que a função de sensoreamento percebe o estado do meio como livre. O autômato pode deixar este nó através de duas arestas, às quais estão associadas operações de leitura sobre um dos canais `navBusy?` ou `phyBusy?`. As operações complementares de escrita sobre os canais são disparadas por transições existentes no autômato que modela o meio físico. Estas transições modelam a detecção por um dos dois mecanismos de sensoreamento de mudanças no estado do meio físico.

Em seguida, a função de sensoreamento do meio deve proceder à imediata notificação das demais tarefas que modelam a estação componente da rede da nova situação do meio. Isto é realizado através do envio de mensagem no canal `busy!` e do registro desta nova situação na variável `csm` (`csm:=BUSY`). Deste modo, o autômato alcança um dos nós `Nav` ou `Phy`, indicando que um dos dois mecanismos de sensoreamento, respectivamente o virtual e o físico, registram o meio como ocupado. Uma vez em qualquer destes dois nós, a detecção da indisponibilidade do meio pelo outro mecanismo leva o autômato para o nó `Both`.

Uma vez no nó `Both`, somente é possível retornar ao nó `Idle` através de visitas aos nós `Nav` ou `Phy`, disparadas por leituras sobre os canais `navIdle` e `phyIdle`. A ordem em que estas leituras ocorrem indicam qual destes dois nós será alcançado primeiro e indica qual dos dois mecanismos detectou inicialmente a liberação do meio físico. Uma vez que a aplicação se encontre em `Nav` ou `Phy` a transição para o nó `Idle` se dará através da visita ao nó `IdleingNAV` ou `IdleingPHY`. Destes nós o autômato poderá finalmente atingir o nó `Idle`, notificando às demais tarefas da estação que o meio se encontra livre. Isto é feito através de mensagem sobre o canal `idle` e da atualização da variável csm (`csm:=IDLE`).

5.3 Modelo do *Point Coordinator*

Para a representação do *Point Coordinator*, usou-se o autômato da figura 5.4.

Para melhor compreensão, este autômato será analisado considerando-se três grupos: o primeiro deles é composto pelos nós `Init`, `InitPCF`, `WaitPIFS`, `MediumBusy` e `SendingBeacon` e compõem o estado no qual o PC busca iniciar um período livre de contenção. O segundo grupo, composto pelos nós `SendingCF_END`, `SendingCF_END$CF_ACK`

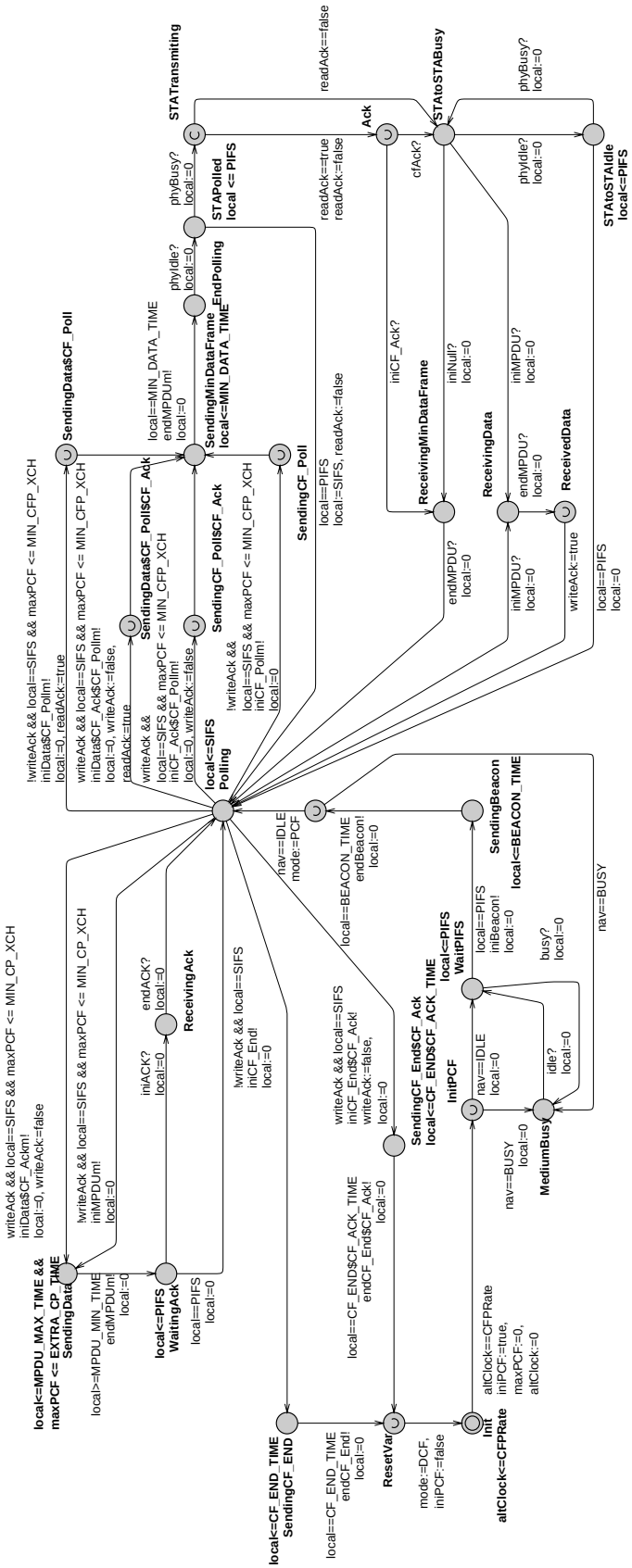

Figura 5.4 *Point Coordinator*

e `ResetVar`, formam o período durante o qual o PC está concluindo um CFP. Os demais nós compõem o CFP propriamente dito.

5.3.1 Início de um CFP

Devido à necessidade de alternância entre os CPs e os CFPs, o PC para iniciar um CFP precisa aguardar pelo menos um período dado pela taxa de repetição do CFP (CFP_Rate). Assim, apenas após a passagem de um CFP_Rate, condição garantida pela associação da invariante `altClock <= CFPRate` do nó `Init` com o guarda `altClock == CFPRate` presente na aresta que deixa este nó, o PC poderá tentar iniciar o CFP e migrar para o nó `InitPCF`.

No nó `InitPCF`, o PC deverá fazer de imediato a verificação do estado do meio físico e caso ele esteja ocupado migrar para o nó `MediumBusy`, onde aguardará até que o meio volte a ficar livre. Caso o meio esteja livre, o autômato migrará para o nó `WaitPIFS`, onde aguardará por um PCF *interframe spacing*. Se durante este período, o meio vir a ficar ocupado o autômato irá para o nó `MediumBusy` (aresta com ação de sincronização `busy?`). Após um PIFS de inatividade, o PC inicia a transmissão de um *Beacon Frame* (aresta com ação de escrita sobre o canal `iniBeacon`) e o autômato passa para o nó `SendingBeacon` e permanece neste nó o tempo necessário para a transmissão do quadro *beacon*. Após este período, garantido pelo invariante associado ao nó e ao guarda associado a aresta de saída, a transmissão do *beacon* é encerrada (aresta com ação de sincronização `endBeacon!`) e o PC passa a operar na PCF.

5.3.2 Encerramento de um CFP

Para encerrar o CFP, o PC deverá aguardar um período SIFS de inatividade do meio no nó `Polling` e em seguida atuar de duas maneiras distintas.

A primeira forma de término do CFP é representada pela migração para o nó `SendingCF_End$CF_Ack` (através de uma transição com ação `iniCF_End$CF_Ack!`). Ela ocorre caso o PC tenha recebido algum MPDU previamente e, portanto, deva enviar um quadro de confirmação Ack (situação indicada através do valor `true` na variável lógica `writeAck`). Neste caso, ele deverá terminar o CFP através do envio de um quadro CF_End+ CF_Ack. O autômato permanece neste nó pelo tempo necessário para a transmissão do quadro de encerramento do PCF, e então retorna para o nó inicial, através de uma travessia pelo nó urgente `ResetVar`, a partir de onde é realizada a reinicialização das variáveis de controle do autômato.

A segunda forma de término do CFP ocorre quando o PC não está com nenhum quadro pen-

dente de confirmação. Neste caso, o autômato migrará para o nó `SendingCF_End` (transição com ação `iniCF_End!`) e permanecerá neste nó o tempo necessário para envio de um quadro de controle do tipo CF_End. Em seguida, o autômato irá voltar para o nó `Init`, reinicializando as variáveis de controle (visita ao nó `ResetVar`).

5.3.3 Comportamento do PC durante um CFP

Durante a duração do CFP, o PC irá realizar uma série de ciclos nos quais ele visitará periodicamente o nó `Polling`. Em cada um destes ciclos, o PC poderá escalonar uma estação e poderá optar por remeter dados para alguma estação.

Na situação em que o PC, detendo o controle do meio físico, resolve enviar dados para uma estação sem escaloná-la, o autômato terá um comportamento que pode ser resumido através de uma travessia pelos seguintes nós: `SendingData`, `WaitingAck` e `ReceivingAck`. A partir do nó `Polling`, o PC poderá atingir o nó `SendingData` através de duas transições: a primeira, com ação de sincronização `iniData$CF_Ackm!` modela a necessidade de o PC responder positivamente a um quadro de dados que lhe tenha sido previamente enviado (variável `writeAck` com valor `true`); a outra aresta modela a inexistência de qualquer pendência de quadros de confirmação, caso em que o PC simplesmente envia dados (transição com ação de sincronização `iniMPDUm!`). A permanência no nó `SendingData` irá variar entre o tempo necessário para enviar o menor quadro de dados (`MPDU_MIN_TIME`) e o tempo necessário para enviar o maior quadro de dados (`MPDU_MAX_TIME`). Em seguida, o PC transita para o nó `WaitingAck` (sinalizando o término do envio do quadro de dados `endMPDUm!`), onde irá aguardar pela recepção de um Ack (transição com ação `iniACK?`) ou até que se passe um PIFS (ocorrência de um *timeout*). Caso receba o Ack, o autômato transitará para o nó `ReceivingAck` e permanecerá neste nó até o fim da recepção do Ack e depois retornará para o nó `Pooling` (transição com ação `endACK?`).

Após a ocorrência de um *timeout*, ou seja, passado um PIFS, se a estação a quem os dados se destina, ainda não enviou um quadro Ack para o PC, o autômato migrará do nó `WaitingAck` diretamente para o nó `Init`, modelando uma situação em que a transmissão de dados não se deu com sucesso.

Na outra situação, o PC irá adotar uma das possíveis formas de escalonar uma estação. Em uma possibilidade de escalonamento, o PC poderá necessitar despachar dados para a estação escalonada, caso em que combinará o quadro de escalonamento com um quadro de dados (CF-Poll+Data). Esta situação é descrita pelo nó `SendingData$CF_Poll`. Em outro caso, O PC poderá ainda necessitar confirmar a recepção de quadro que lhe foi destinado antes do último

ciclo de comunicação (anterior ao último SIFS). Neste caso, deverá combinar o quadro de escalonamento com um quadro CF-Ack (CF-Poll+CF-Ack). O nó `SendingCF_Poll$CF_Ack` modela este segundo caso. O PC poderá também necessitar tanto confirmar a recepção de dados previamente recebidos, quanto enviar dados para a estação escalonada. Nesta situação, o quadro a ser utilizado será um CF-Poll+CF-Ack+Data. Esta opção de escalonamento é representada pelo nó `SendingData$CF_Poll$CF_Ack`. Finalmente, o PC pode simplesmente escalonar a estação, sem que seja necessário enviar nenhum quadro de confirmação ou mesmo despachar dados para a estação escalonada fazendo uso de um quadro CF-Poll. Este caso está modelado através do nó `SendingCF_Poll`.

Quando o PC escalona uma estação seu comportamento é dependente do tipo de quadro utilizado para o escalonamento. Deste modo, o início de envio do CF-Poll isolado é representado através da transição com ação de sincronização `iniCF_Pollm!` que leva o autômato do nó `Polling` para o nó `SendingCF_Poll` e daí, de imediato, para `SendingMinDataFrame`. Por sua vez, o envio do CF-Poll combinado com um quadro de dados é modelado através da transição com ação `iniData$CF_Pollm!` que leva o autômato nó `Polling` para o nó `SendingData$CF_Poll`, de onde ele alcançará imediatamente o nó `SendingMinDataFrame`. Ambas as transições somente estarão habilitadas, caso o PC não tenha consumido dados enviados por uma estação no ciclo anterior e, portanto, não necessite enviar um quadro de confirmação (situação modelada pela expressão `writeAck` presente nos dois guardas).

Caso o PC tenha consumido dados enviados por alguma estação, a variável `writeAck` terá valor `true`. Neste caso, somente estarão habilitadas as transições que representem o envio combinado de um CF-Ack com um CF-Poll. Assim, o autômato irá escolher aleatoriamente entre transitar para o nó `SendingCF_Poll$CF_Ack` (através da transição `iniCF_Ack$CF_Pollm!`) ou transitar para o nó `SendingData$CF_Poll$CF_Ack` (através da transição `iniData$CF_Ack$CF_Pollm!`), modelando a possibilidade do PC combinar o Ack e o Poll com dados que necessitem ser despachados para a estação escalonada. Novamente, nos dois casos, o autômato irá prosseguir de imediato para o nó `SendingMinDataFrame`.

O autômato permanece no nó `SendingMinDataFrame` o tempo necessário para o envio do menor quadro de dados, visto que todos os quadros de controle da PCF são embutidos em quadros de dados de tamanho fixo e equivalente ao menor quadro de dados. Em seguida, o PCF descarta a notificação de mudança do estado do meio físico para livre (transição com ação de sincronização `phyIdle?`) gerada em decorrência do término da transmissão do seu quadro de escalonamento e passa a aguardar que o meio volte a ficar ocupado ou que um

PIFS ocorra. Caso o meio não volte a ficar ocupado em no máximo um PIFS, significa que a estação escalonada, por algum motivo, não fez uso da sua vez de transmitir. Isto pode ocorrer, por exemplo, por falha na transmissão do quadro de escalonamento. Neste caso, ocorre um *timeout* e o PC retoma o controle do meio, retornando ao nó `Polling` (aresta com guarda `local==PIFS` com origem no nó `STAPolled` e destino no nó `Polling`). Em seguida, de imediato, o PC volta a escalonar o meio, pois já se passou um período entre quadros maior que um SIFS sem que o meio tivesse sido utilizado. Isto é garantido pela combinação da atribuição `local:=SIFS` existente na aresta com o invariante `local<=SIFS` associado ao nó `Polling`.

Caso o meio volte a ficar ocupado, situação detectada pela transição com ação de sincronização `phyBusy?`, o PC move-se para o nó `STATransmiting`, que representa que a estação escalonada passou a fazer uso do meio. Caso o PC necessite receber uma confirmação referente ao quadro que despachou para a estação escalonada, o autômato migrará de imediato para o nó `Ack` (aresta com guarda `readAck`), ou, em caso contrário, para o nó `NoAck`(aresta com guarda `!readAck`).

Uma vez no nó `Ack`, o autômato do PC poderá migrar para o nó `ReceivingCFAck` através da transição com ação de sincronização `iniCF_Ack?` e daí para o nó `ReceivingMinDataFrame`, que modela a recepção de um quadro CF-Ack puro, sem combinação com nenhum outro. Caso o quadro venha combinado, a fração CF-Ack é consumida pelo PC e o autômato transfere-se para o nó `DataTo` (transição com leitura `cfAck?`). Este nó modela a possibilidade dos dados transmitidos pela estação escalonada se destinar ao próprio PC ou a uma outra estação. No primeiro caso, o PC consome os dados transmitidos, o que é representado pela ativação da aresta com ação de sincronização `iniMPDU?`, que leva o autômato para o nó `ReceivingDataCFAck` e daí para o nó `ReceivingData`. No caso da comunicação destinar-se a outra estação, os dados serão consumidos pela estação destino e o autômato permanecerá no nó `DataTo`, monitorando o estado do meio, até que este fique livre, quando ele migrará para o nó `STAtoSTAIdle` (ativação da aresta com ação de sincronização `phyIdle?`).

Caso não tenha havido o envio de dados para a estação escalonada, o autômato segue a aresta com guarda `!readAck` e se encontrará no nó `NoAck`. A partir deste nó, o autômato poderá seguir através de uma das arestas com ação de sincronização `phyIdle?`, `iniNull?` ou `iniMPDU?`. A primeira aresta descreve a transmissão de um quadro pela estação escalonada que não se destina ao PC. Deste modo, o PC simplesmente percebe que o meio se tornou ocupado e o autômato migra para o nó `STAtoSTAIdle`. A segunda aresta modela a transmissão de um quadro Null pela estação, indicando que ela optou por não transmitir dados neste ciclo.

Por fim, a terceira aresta exibe a transmissão de dados destinados ao próprio PC.

No nó `STAtoSTAIdle`, o autômato ficará alternando entre os nós `STAtoSTABusy` e `STAtoSTAIdle`, de acordo com o estado do meio (arestas com ações de sincronização `phyIdle?` e `phyBusy?`), até que o período de inatividade do meio seja igual a um PIFS, o que indica a conclusão do processo de comunicação entre as duas estações ou a ocorrência de um *timeout*. Então, o PC retomará o controle do meio e retornará para o nó `Polling`, escalonando de imediato outra estação. Este escalonamento imediato, deve-se mais uma vez, à passagem de um período de inatividade superior a um SIFS desde a última utilização do meio.

Caso tenha ocorrido a ativação da aresta com ação de sincronização `iniNull?` ou `iniCFAck?`, o autômato migrará para o nó `ReceivingMinDataFrame`, e permanecerá neste nó até o término da transmissão do quadro Null ou CF-Ack. Ao final da transmissão, o autômato retornará ao nó `Polling` (ativação da aresta com ação de sincronização `endMPDU?`), onde permanecerá por um SIFS antes de voltar a utilizar o meio.

Caso os dados tenham sido destinados ao próprio PC (ativação de uma das duas arestas com ação de sincronização `iniMPDU?`), o autômato permanecerá no nó `ReceivingData` até a conclusão da recepção do quadro de dados (aresta com ação de sincronização `endMPDU?`). Ao fim da recepção, o autômato migrará para o nó `ReceivedData`, de onde alcançará o nó `Polling`, registrando a necessidade do PC enviar uma confirmação de recepção de dados no próximo quadro (aresta com atribuição `writeAck:=true`).

5.4 Estações na PCF

O autômato que foi utilizado para representar o comportamento das estações quando operam durante o período livre de contenção está representado na figura 5.5.

Durante o PCF a estação apenas deixará o nó inicial (`Init`) em uma das duas situações seguintes. A primeira delas acontece quando a estação consome um MPDU, que tenha sido a ela destinado (descrita na seção 5.4.1). A segunda situação acontece quando a estação foi escalonada pelo PC (seção 5.4.2). A seguir, discutiremos detalhadamente o comportamento do autômato nos dois casos apresentados.

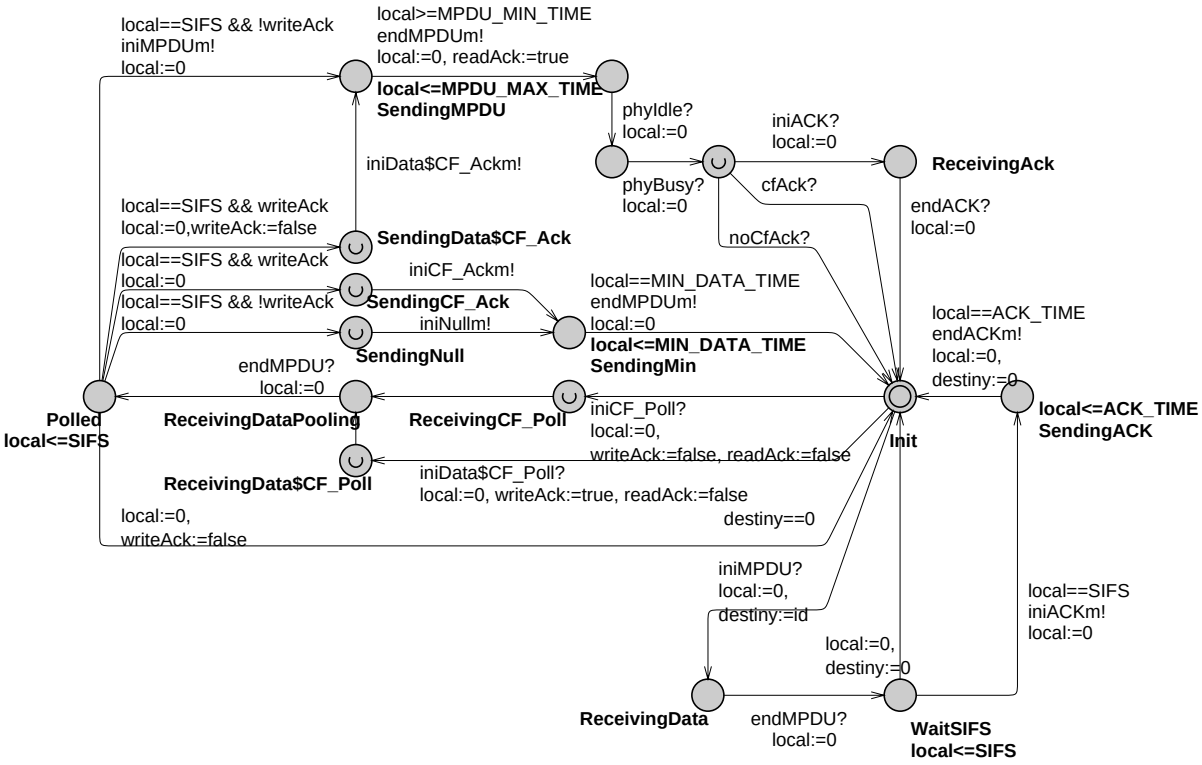

Figura 5.5 Estação operando sob a PCF

5.4.1 Consumo de MPDU

O primeiro comportamento possível consiste no autômato deixar o nó inicial (`Init`) ao consumir um quadro de dados (transição com ação `iniMPDU?`). Neste caso, a estação se comporta de modo similar à tarefa de leitura das estações no DCF básico, ou seja, a estação irá permanecer no nó `ReceivingData` até que seja concluída a transmissão dos dados que lhe foram destinados. Assim que a transmissão do MPDU for concluída (transição com ação de sincronização `endMPDU?`), o modelo transitará para o nó `WaitSIFS` e permanecerá neste nó por no máximo um SIFS (invariante `local<=SIFS` do nó `WaitSIFS`). Durante este período poderá optar por retornar ao nó `Init` ou então após exatamente um SIFS transitar para o nó `SendingAck`. O primeiro caso modela o não envio de um quadro de confirmação decorrente de uma falha na transmissão. O segundo modela o envio do quadro de confirmação, após o que o autômato retorna ao nó `Init` (transição com ação de sincronização `endACKm!`).

5.4.2 Estação Escalonada

No segundo comportamento possível, a estação pode deixar o nó `Init` através de duas transições. A primeira delas é ativada através de uma ação de sincronização `iniData$CF_Poll?` e modela o início da recepção de um quadro combinado CF-Poll+Dados destinado à estação. Nesta transição é atribuido o valor `true` à variável local `writeAck`, para indicar que a estação deverá enviar posteriormente um quadro de confirmação (`writeAck:=true`). A segunda transição é ativada através de uma ação de sincronismo `iniCF_Poll?` e representa que a estação escalonada recebeu apenas um quadro CF-Poll, sem nenhum dado associado. Neste caso, é atribuído à variável `writeAck` o valor `false`.

Em qualquer uma das duas situações, a transição alcançará de imediato o nó `ReceivingDataPooling` (no primeiro caso através do nó urgente `ReceivingCF_Poll` e no segundo caso através do nó urgente `ReceivingData$CF_Poll`) e aí permanece até o fim da recepção do quadro de dados (CF-Poll ou CF-Poll+Data). Ao fim da transmissão, representada pela ação de sincronização `endMPDU?`, a estação irá migrar para o nó `Polled`, onde deverá aguardar pela passagem de um SIFS. Em seguida, o caminho a ser tomado é dependente da necessidade ou não da estação confirmar a recepção do quadros de dados ao PC.

Caso a estação deva confirmar a recepção do quadro de dados (variável `writeAck` com valor `true`), a estação poderá deixar o nó `Polled` através de duas transições. Na primeira delas, a estação escalonada combina o quadro Ack com dados a serem enviados (transição com ação de sincronização `iniData$CF_Ackm!`) e migra para o nó que modela o envio de dados por parte da estação (`SendingMPDU`). Caso a estação não possua nenhum dado a enviar, o

autômato migrará para o nó SendingCF_Ack, de onde alcançará o nó SendingMin através de uma transição com ação de sincronização iniCF_ACKm!. Esta transição modela o início da transmissão de um quadro de confirmação do modo CFP (CF-Ack).

Caso a estação não precise enviar nenhum quadro de confirmação, pois não recebeu dados (variável writeAck com valor false), ela poderá se comportar de dois modos distintos ao ser escalonada. No primeiro, a estação decide que não irá enviar nenhum dado para o PC ou para outra estação. Assim, a estação responde ao escalonamento enviando um quadro Null, situação modelada através da transição com ação iniNullm! No segundo, a estação decide enviar dados e inicia a transmissão de um MPDU (transição com ação de sincronização iniMPDUm!) e o autômato migra para o nó SendingMPDU.

Nos dois casos em que a estação não envia dados, as transições ativadas levam o autômato para o nó SendingMin. O autômato permanece neste nó durante o tempo necessário para o envio do menor quadro de dados (MIN_DATA_TIME). Este é o tempo utilizado para a transmissão dos quadros Null e CF-Ack, que são embutidos em um quadro de dados de tamanho mínimo.

A partir do nó SendingMin o autômato migra para o nó inicial, sinalizando o término da transmissão do quadro (ação de sincronização endMPDUm! associada à transição).

Nas situações em que a estação realiza a transmissão de dados (isoladamente ou combinado com o quadro CF-Ack), o autômato permanecerá no nó SendingMPDU o tempo necessário à transmissão do MPDU (entre MPDU_MIN_TIME e MPDU_MAX_TIME), de onde aguardará o quadro de confirmação. Para isto, o autômato consome duas sinalizações, a primeira gerada em função do término do quadro enviado por ela própria (notificação no phyIdle?) e a segunda gerada em função do meio ter voltado a ficar ocupado devido à transmissão de um novo quadro por uma outra estação ou pelo PC (notificação através de phyBusy?).

A seguir, o autômato irá detectar o tipo de quadro que foi transmitido pela estação escalonada ou pelo PC. Caso o quadro transmitido seja um quadro de confirmação enviado por uma outra estação, a estação migrará para ReceivingAck e daí volta para o nó Init ao final da leitura deste quadro (transição com ação de sincronização endACK?). Caso o quadro transmitido pelo PC ou pela outra estação seja um quadro CF-Ack isolado ou embutido em outros quadros (CF-Poll+CF-Ack, Data+CF-Ack ou Data+CF-Poll+CF-Ack), a estação percebe que o quadro recebido é um quadro de confirmação e ativa a aresta com ação de sincronização cfAck?, indicando que a transmissão foi feita com sucesso. Caso a leitura seja de qualquer outro tipo de quadro, a estação simplesmente volta para o nó inicial (transição com ação noCfAck?) e a trasmissão termina sem sucesso.

5.5 Alternância entre o CP e o CFP

A alternância entre o CP e o CFP está modelada nos autômatos que representam as tarefas que atuam durante o CFP (PC e Estações na PCF - figuras 5.4 e 5.5).

No PC, o controle da alternância entre as funções de coordenação foi realizado através da declaração de duas variáveis locais do tipo relógio, `altClock` e `maxPCF`. A variável `altClock` é utilizada para controlar a taxa de repetição do CFP. Assim, o invariante presente no nó `Init` (`altClock<=CFPRate`) e o guarda associado à aresta que conecta este nó ao nó `InitCFP` (`altClock==CFPRate`) garantem que o PC somente pode iniciar a sua tentativa de controlar o meio, através do início de um CFP, após a passagem de um período CFP_Rate. No momento da transição, `altClock` é reiniciado, de forma a passar a contabilizar um novo período de duração igual a CFP_Rate para o início do próximo CFP.

A segunda variável, `maxPCF`, é usada para controlar a duração do PCF. Este relógio é reiniciado no momento em que o PC passa a tentar controlar o meio e é verificado nas transições que partem do nó `Pooling` e modelam o escalonamento de estações e as transferências de dados do PC para as estações. Nestas transições, é verificado se o tempo disponível até o fim do CFP é suficiente para uma sequência atômica de transferência do menor quadro de dados (`maxPCF <= MIN_CP_XCH`, no caso de uma troca de dados similar à do CP ou `maxPCF <= MIN_CFP_XCH`, no caso de uma troca de dados típica do CFP).

No autômato que modela o comportamento de uma estação na PCF, o controle de coexistência do CFP é baseado em uma variável relógio local chamada `cpfDuration`. Esta variável é reiniciada a cada início de CFP, condição detectada pela recepção de um quadro *beacon* (transição no nó `Init` com ação de reinicialização `cpfDuration:=0`). Assim, esta variável é utilizada para controlar o tamanho de um quadro de dados, de modo que o tempo gasto na sua transmissão não extrapole o tempo disponível para o PCF (invariante com fórmula `cpfDuration <= EXTRA_CP_TIME` associada ao nó `SendingMPDU`).

5.6 Meio Físico

No modelo proposto foi inserido um autômato, apresentado na figura 5.6, para modelar o meio físico. Esta representação foi necessária para modelar a possibilidade de ocorrência de colisão, ou seja, duas estações iniciarem simultaneamente a utilização do meio físico, e para representar a possibilidade de um mesmo quadro ser destinado a duas estações distintas, como é o caso dos quadros de confirmação combinados, utilizados durante o PCF. Para este segundo requisito, o

autômato deve separar a informação de Ack das demais informações do quadro, para que cada modelo de estação possa consumir a parte que lhe é destinada.

Este autômato é composto pelos seguintes nós:

- `Init`: modela o estado inicial;
- `RcvPacket`: modela o estado durante o qual um pacote está sendo lançado no meio;
- `TransPacket`: modela o instante em que o meio verifica se é possível repassar o pacote recebido à estação destino ou se irá descartá-lo por ocorrência de colisão;
- `Collision`: representa o período em que o meio que teve mais de um pacote simultaneamente transmitido aguarda pelo fim da transmissão dos demais pacotes;
- `NotCollision`: simboliza a ocorrência da transmissão de um pacote sem colisão;
- `TransCFPool`, `TransDataCFPool`, `TransCFAckCFPool`, `TransDataCFAckCFPoll`, `TransDataCFAck`, `TransCFAck`, e `TransNull`: representam o repasse feito pelo meio físico para uma estação receptora de um dos quadros de controle do PCF;
- `Discard`: indica o descarte de quadros pelo meio em caso de colisão, visto que as estações estarão inabilitadas de reconhecer o conteúdo dos quadros transmitidos simultaneamente;
- `SplitAck` : modela a divisão da informação de Ack presente em um quadro do tipo CF-End+Cf-Ack;
- `NotifyIdle1` e `NotifyIdle2`: modela estados de conclusão do PCF e que, portanto, necessitam de reinicialização do mecanismo de sensoreamento virtual das estações;
- `WaitBroadFrame`: representa o consumo pelo meio de um quadro de *broadcast* e que, portanto, não se destina a nenhuma estação em particular;
- `EndFrame`: representa o nó onde o meio detecta se a notificação é de início ou de término de quadro. Caso seja de término, um sinal de meio livre é gerado para os mecanismos de sensoreamento do meio.

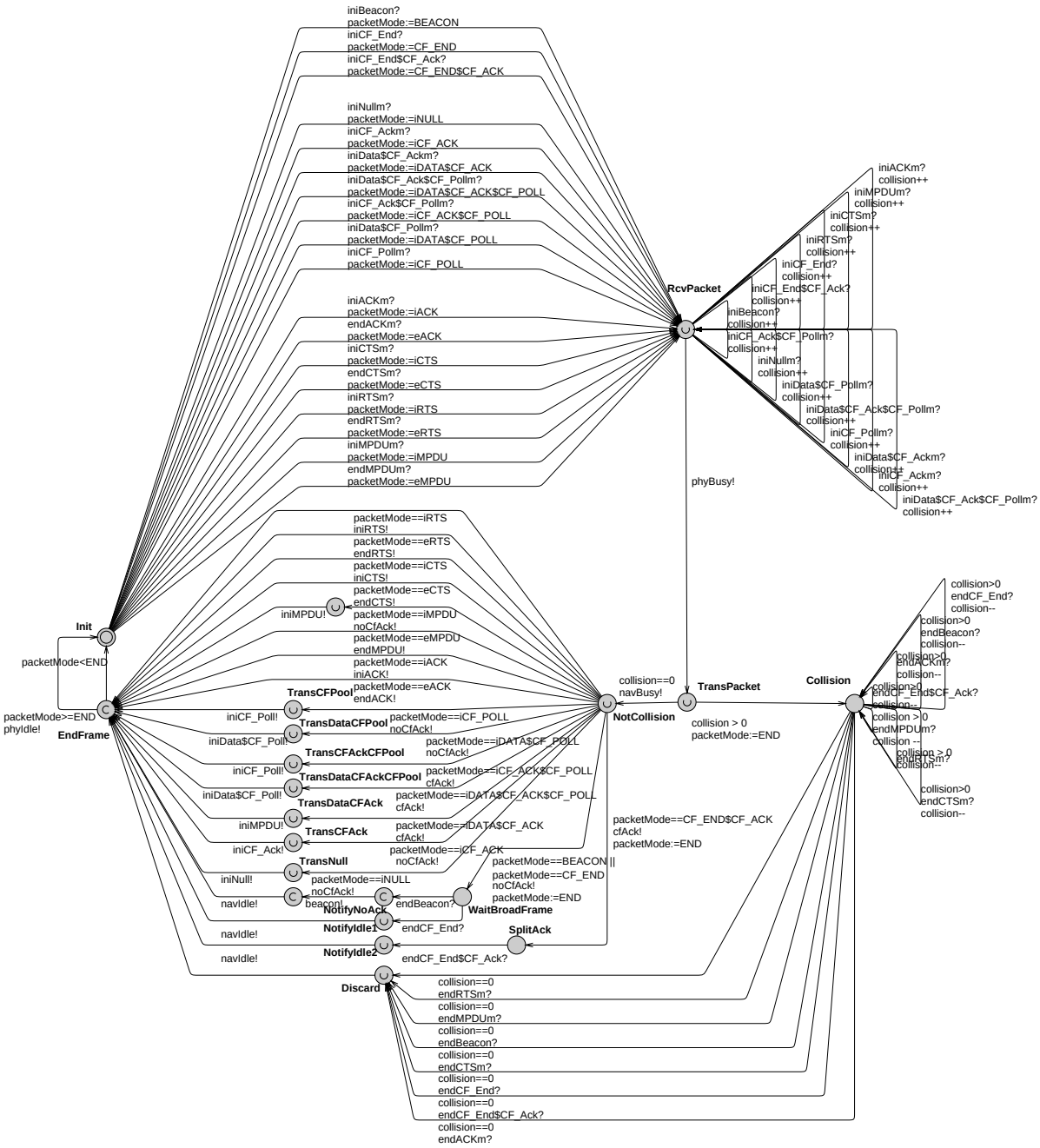

Figura 5.6 Meio físico

O autômato permanece no nó `Init` até uma estação iniciar ou finalizar a transmissão de um quadro qualquer. Quando isto ocorrer, em função de qual quadro esteja sendo transmitido, uma das arestas que conectam este nó ao nó `RcvPacket` será ativada e a variável inteira local `packetMode` registrará qual o tipo de evento que ativou a transição.

Embora o tempo não possa passar no nó `RcvPacket`, visto que este nó é urgente, o modelo permite intercalações entre os diversos autômatos que compõem o sistema, o que faz com que outras mensagens enviadas para o meio, simultaneamente ao evento que ativou a transição, disparem uma das arestas com origem e destino no próprio nó `RcvPacket`, incrementando a variável inteira local `collision`.

Mas uma vez sem consumo de tempo, o autômato migra para o nó `TransPacket`. Neste nó o autômato deverá tomar um dos dois caminhos: transitar para o nó `NotCollision`, se nenhuma colisão foi registrada (transição com guarda `collision == 0`); ou para o nó `Collision`, em caso contrário (transição com guarda `collision > 0`).

No nó `Collision` os registros de término da transmissão dos quadros vão sendo descartados através das transições existentes neste nó, até o descarte do último quadro, quando o autômato migrará para o nó `Discard` (guarda `collision==0` associados às transições que conectam os dois nós). Imediatamente o autômato migrará para o nó `EndFrame` e daí, também sem consumo de tempo, para o nó `Init`. Esta última transição se dará com a geração de sinal de meio livre, caso a notificação repassada para as estações seja de fim de quadro ou sem esta notificação, caso o sinal seja de início de transmissão de quadro.

No nó `NotCollision`, caso a transmissão seja de um quadro típico da DCF, o autômato simplemente repassará para um dos modelos de tarefa de recepção o evento de início ou término de transmissão de quadro, de acordo com o indicado na variável `packetMode`, e vai para o nó `EndFrame`.

Para os quadros que são transmitidos durante o período livre de contenção, este autômato é responsável por modelar mais um comportamento em particular. Na PCF, quadros de confirmação de recepção relativos a dados despachados no ciclo anterior, podem ser combinados com quadros de escalonamento e de dados destinados a uma estação distinta daquela a que o quadro de confirmação se destina. Além disso, uma estação que aguarde um quadro de confirmação deve estar apta a perceber que o quadro que foi transmitido no novo ciclo não traz embutido a confirmação esperada, mesmo quando este quadro de escalonamento ou de dados não se destine a ela. Para modelar este comportamento, utilizou-se de dois canais de comunicação, `noAck` e `cfAck`. Estes canais são utilizados para que o meio possa indicar se o quadro transmitido é ou não um quadro da família dos quadros de confirmação, ou seja, se traz embutido a confirmação de recepção de dados do ciclo anterior. Se o quadro pertencer à família dos quadros

de confirmação (CF-Ack+CF-Poll, CF-Ack+Data, CF-Poll+CF-Ack+Data, CF-End+CF-Ack) uma operação de escrita no canal `cfAck` é realizada. Caso o quadro não pertença à família dos quadros de confimação (CF-Poll, CF-Poll+Data, Null, CF-End, Beacon, MPDU) uma operação de escrita sobre o canal `noAck` acontece. No caso do quadro CF-Ack, como não há a combinação de confirmação com outro quadro, nenhuma notificação é necessária.

Assim, para os quadros típicos do modo PCF, a lógica seguida é a seguinte: os quadros com informação de confirmação embutidos são divididos em duas notificações: uma disparada por ação de escrita no canal `cfAck` e a outra dependente do tipo do quadro que está sendo enviado. Assim, uma ação de sincronização `iniCF_Ack$CF_Pollm$` gera duas ações de sincronização distintas, `cfAck!` e `iniCF_Pollm!`.

Finalmente, os quadros que não são destinados a nenhuma estação em particular (os quadros de *broadcast* Beacon, CF-End e CF-Ack+CF-End) são consumidos pelo meio físico. Quando necessário, o autômato trata de notificar aos mecanismos de sensoreamento que o meio voltou a ficar livre. Além disso o autômato realiza o desmembramento da informação de confirmação no caso de quadros CF-Ack+CF-End.

CAPÍTULO 6

Verificação Formal da Camada de Acesso ao Meio do IEEE 802.11

O senso de realidade é vital na lógica.

—BERTRAND RUSSELL (Matemático e Filósofo Inglês)

Este capítulo descreve o processo de verificação formal da camada de acesso ao meio do protocolo IEEE 802.11. Na seção 6.1 são descritas as hipóteses assumidas no funcionamento da rede e são apresentadas as configurações de redes utilizadas para a verificação. A seção 6.2 descreve qual foi o ambiente computacional no qual as verificações foram realizadas e a seção 6.3 apresenta as fórmulas que modelam as propriedades submetidas ao verificador e apresenta o resultado retornado pela ferramenta.

6.1 Configurações e Hipóteses

Para a verificação das propriedades, foram considerados o funcionamento isolado de cada uma das funções de coordenação do uso do meio, e, em seguida, o funcionamento conjunto das duas funções.

Durante a verificação, aumentou-se gradualmente a complexidade dos cenários através do aumento do número de estações. Para este fim foram criados uma série de processos a partir dos modelos descritos no capítulo 5. Os processos construídos a partir dos autômatos que modelam a tarefa de envio de mensagens na DCF foram nomeados como `dcf1`, `dcf2`, `dcf3` e `dcf4`. Os processos que representam as funcões de sensoreamento do meio e do PC foram chamados de `dcfcs1`, `dcfcs2`, `dcfcs3`, `dcfcs4` e `pccs`. Os processos de recepção de mensagens foram declarados como `rdcf1`, `rdcf2`, `rdcf3` e `rdcf4`. O *Point Coordinator* é modelado pelo processo `pc` e o meio físico é modelado por `phy`. Finalmente os processos `pcf1`, `pcf2`, `pcf3` e `pcf4` representam o comportamento das estações escalonáveis durante um CFP.

Estes processos foram combinados paralelamente em sistemas, conforme as configurações apresentadas na tabela 6.1. As configurações DCF 1, DCF 2, DCF 3 e DCF 4 modelam situações em que a rede é composta apenas por estações que operam sob a DCF (DCF exclusivo). Já

as configurações PCF 1, PCF 2, PCF 3 e PCF 4 modelam redes compostas por um *Point Coordinator* e estações que só transmitem quando escalonadas pelo PC (PCF exclusivo). Finalmente, DCF1+PCF1, DCF2+PCF2, DCF3+PCF2 modelam redes em que estações que operam sob as regras da DCF coexistem com estações escalonáveis pelo PC.

Tabela 6.1 Configurações utilizadas nas Verificações

	Configuração
DCF 1	Composta pelos processos `dcf1`, `dcfcs1`, `rdcf1` e `phy`
DCF 2	Composta pelos processos `dcf1`, `dcfcs1`, `dcf2`, `dcfcs2`, `rdcf1` e `phy`
DCF 3	Composta pelos processos `dcf1`, `dcfcs1`, `dcf2`, `dcfcs2`, `dcf3`, `dcfcs3`, `rdcf1` e `phy`
DCF 4	Composta pelos processos `dcf1`, `dcfcs1`, `dcf2`, `dcfcs2`, `dcf3`, `dcfcs3`, `dcf4`, `dcfcs4`, `rdcf1` e `phy`
PCF 1	Composta pelos processos `pc`, `pccs`, `pcf1` e `phy`
PCF 2	Composta pelos processos `pc`, `pccs`, `pcf1`, `pcf2` e `phy`
PCF 3	Composta pelos processos `pc`, `pccs`, `pcf1`, `pcf2`, `pcf3` e `phy`
PCF 4	Composta pelos processos `pc`, `pccs`, `pcf1`, `pcf2`, `pcf3`, `pcf4` e `phy`
DCF1 + PCF1	Composta pelos processos `pc`, `pccs`,`dcf1`, `dcfcs1`, `rdcf1`, `pcf1` e `phy`
DCF2 + PCF2	Composta pelos processos `pc`, `pccs`, `dcf1`, `dcfcs1`, `dcf2`, `dcfcs2`, `rdcf1`, `rdcf2`, `pcf1`, `pcf2` e `phy`
DCF3 + PCF2	Composta pelos processos `pc`, `pccs`, `dcf1`, `dcfcs1`, `dcf2`, `dcfcs2`, `dcf3`, `dcfcs3`, `rdcf1`, `rdcf2`, `rdcf3`, `pcf1`, `pcf2` e `phy`

Em nenhuma das configurações utilizou-se mais de um PC. Isto significa que foi afastada a possibilidade de colisão de quadros *beacons*. Assumir esta hipótese implica em garantir que o PC não entrará em *backoff* devido à interferência de outros PCs existentes na mesma BSA e que utilizam camadas físicas com as mesmas características.

6.2 Ambiente de Verificação

Para a verificação utilizou-se a versão 3.4.11 do UPPAAL. A configuração do computador, onde a ferramenta foi executada, era composta por um PC, processador Intel Celeron com velocidade de 2.3 GHz e 1GB de memória RAM. As medições de tempo de CPU e uso de memória foram

realizadas pela ferramenta Process Explorer (Rus05).

Durante a verificação, fêz-se uso da ferramenta de linha de comando `verifyta` do UPPAAL. Esta ferramenta permite a execução da verificação em lotes das diversas propriedades armazenadas em um único arquivo. Todas as configurações passíveis de serem feitas na GUI (*Graphical User Interface*) podem também ser feitas na ferramenta através de parâmetros de linha de comando. Na verificação foram utilizados parâmetros que diminuem a utilização de memória em detrimento do tempo de verificação. Esta configuração foi adotada, em função da explosão de espaço de memória causado pela composição paralela dos autômatos existentes no modelo, que se mostrou como o maior fator restritivo das verificações. Os parâmetros utilizados foram: -U, que indica que o espaço de estados é representado através de *minimal constraint graphs*; e -S2, que indica o uso de uma técnica agressiva para a redução do espaço de estados.

6.3 Propriedades Verificadas

Para a verificação, buscou-se identificar um conjunto de propriedades úteis aos projetistas de protocolos e aplicações. Estas propriedades buscam evidenciar algumas propriedades de *safety*, *liveness* e *timeliness* associadas ao protocolo.

Propriedade 1 (Ausência de *deadlock*)**.**

Esta propriedade foi descrita em $\mathscr{L}_s$ através da fórmula:

```
A[]!deadlock
```
(6.1)

Esta propriedade é uma propriedade do tipo *deadlock-freeness* e foi verificada para configurações com estações que operam apenas no modo DCF, para configurações com estações no modo PCF exclusivo, e para o funcionamento conjunto das duas funções de controle de acesso ao meio.

Propriedade 2 (Acesso Exclusivo ao Meio)**.**

Nesta propriedade, deseja-se verificar se o meio físico é utilizado por apenas uma das estações participantes da rede por vez. Diversas fórmulas poderiam ter sido utilizadas para a verificação desta propriedade. Neste trabalho, foi considerado que quando duas estações acessam simultaneamente o meio físico, tem-se uma situação de colisão.

Assim, foi expressa em $\mathscr{L}_s$ a condição do estado de colisão do meio físico ser sempre inalcançável, através da seguinte fórmula:

```
A[]!phy.Collision                                    (6.2)
```

Onde o processo `phy` é instanciado a partir do autômato que modela o meio físico, apresentado na figura 5.6.

Esta é uma propriedade de segurança, pois define um comportamento que o protocolo deve garantir: o acesso ordenado ao meio físico.

Exceto para a configuração DCF 1, a verificação falhou para todas as configurações de DCF exclusivo e conjuntas. Para a configuração DCF 1 a propriedade se mantém, pois, existindo apenas uma estação na rede, não existe a possibilidade da ocorrência de colisão.

A verificação teve sucesso para as configurações PCF exclusivo. A partir deste resultado, inferiu-se a necessidade de se verificar se durante a operação conjunta, quando a função de acesso é a PCF, o acesso exclusivo se mantém. Esta propriedade está especificada a seguir.

Propriedade 3 (Acesso Exclusivo ao Meio no modo PCF)**.**

Esta propriedade descreve que, durante a operação no modo PCF, apenas uma estação acessa o meio físico por vez:

```
A[](mode==PCF imply !phy.Collision)                  (6.3)
```

Para esta propriedade, foi utilizada uma variável global que registra o modo de operação e é atualizada pelo PC, ao entrar e sair de um CFP. Esta propriedade se manteve para todas as configurações fornecidas, demonstrando que durante um PCF não ocorrem colisões. Como a propriedade anterior, esta também é uma propriedade de *safety*, possuindo, portanto, a mesma estrutura sintática que aquela.

Propriedade 4 (*Livelock* de *backoff*)**.**

Esta propriedade descreve que um procedimento de *backoff* que tenha sido iniciado por uma estação, pode, em determinadas condições, nunca alcançar o seu término, ou seja, que não existe garantia de que uma estação que inicie um procedimento de *backoff*, venha um dia a sair dele. Isto deve-se, por exemplo, ao fato da estação que esteja em *backoff* poder ter o seu relógio de *backoff* bloqueado repetidamente infinitamente, devido a transmissões originadas por outras estações.

Para a demonstração desta propriedade, foi especificada em $\mathscr{L}_s$ a sua negação:

```
dcf1.InitBackoff-->dcf1.EndBackoff                   (6.4)
```

Esta fórmula descreve que um procedimento de *backoff* que tenha sido iniciado sempre chegará ao final. O processo `dcf1` é uma instância do modelo apresentado na figura 5.2.

Para a configuração DCF 1, o verificador indicou que a propriedade submetida é verdadeira, indicando não existir, para a configuração em questão, *livelock* de *backoff*. Nesta configuração, o relógio de *backoff* não pode ser infinitamente bloqueado pela transmissão de uma outra estação, simplesmente pelo fato desta outra transmissão não poder ocorrer, devido a ausência de uma outra estação transmissora. Isto justifica o porquê da propriedade original ser falsa para esta configuração

Para as demais configurações DCF exclusivo e para todas as configurações de operação conjunta, a ferramenta indicou ser falsa a propriedade submetida, demonstrando que a propriedade original era verdadeira. Não foram feitas verificações desta propriedade para o modo PCF exclusivo, visto que, nestas configurações o processo (`dcf1`) e o estado envolvidos são inexistentes (`InitBackoff` e `EndBackoff`).

Propriedade 5 (*Livelock* de colisão)**.**

Não existe garantia de que uma estação que deseje transmitir, mesmo que o procedimento de *backoff* seja concluído, o consiga fazer com sucesso, pois a estação pode infinitamente entrar em colisão. Para verificar esta propriedade, foi necessário se eliminar a possibilidade de uma estação entrar em *backoff*, por qualquer outro motivo, que não fosse devido à ocorrência de colisão. Isto pode acontecer, por exemplo, devido a erros encontrados no campo de verificação de quadros transmitidos, que podem levar ao não envio de quadros ACK ou CTS para as estações fontes por parte das estações destino. A ausência destes quadros de confirmação levam à ocorrência de *timeout* e a conseqüente entrada em *backoff*, sem que tenha havido colisão. Assim para evitar esta ocorrência, indesejada para esta verificação, criou-se uma constante lógica global `noiseOnChan` e inseriu-se esta constante como guarda nas duas transições do modelo da tarefa de recepção de mensagens da DCF (subseção 5.2.1), que modelam a recepção de quadros com falhas (transições do nó `WaitSIFS` para o nó `Init` e do nó `ReceivedMPDU` para o nó `Init` - fig. 5.1). Deste modo, para o modelo das tarefas de envio de mensagens na DCF, as transições que dão acesso aos nós `NoCTS` e `NoAck` somente poderão acontecer caso o quadro CTS ou ACK não sejam recebidos devido à ocorrência de colisão na transmissão do quadro RTS ou MPDU. A propriedade foi especificada em $\mathscr{L}_s$, como:

```
dcf1.EndBackoff --> dcf1.Transmited
```
(6.5)

Esta fórmula, similar à da propriedade anterior, indica que se o autômato chegar ao nó que modela o final do procedimento de *backoff* (nó `dcf1.EndBackoff` fig. 5.2), ele, em algum

momento futuro, estará no nó que indica que a mensagem foi transmitida com sucesso, ou seja, que não ocorreu colisão.

Ao submeter esta propriedade para verificação sob as configurações do modo DCF exclusivo e operação conjunta, a propriedade falha para todos os casos, exceto para a configuração DCF 1, onde a possibilidade de colisão inexiste. De forma similar à propriedade anterior, não há sentido em se verificar esta propriedade para as configurações do modo PCF exclusivo.

Propriedade 6 (*Timely Access*)**.**

Esta propriedade demonstra a prioridade do PC no acesso ao meio e define qual o tempo máximo para o PC assumir o controle do canal de comunicação sempre que o PC desejar. Esta é uma propriedade fundamental para a definição das características de tempo-real do canal de comunicação.

Para realizar a verificação desta propriedade foi declarada uma variável relógio (`clock maxPCF;`) local ao modelo do AP (fig. 5.4). Esta variável armazenará o tempo que se passa a partir do instante em que o AP resolve tomar o controle do meio, representado pelo nó `InitPCF`, até o momento em que efetivamente inicia a PCF (nó `Polling`).

O tempo máximo para a PCF iniciar, a partir do instante em que o PC resolve controlar o acesso ao meio, é dado por dois períodos PIFS: um primeiro para uma eventual colisão e o segundo para o acesso prioritário do PC. Estes são somados ao tempo de transmissão do quadro *beacon*, que inicia o CFP, e ao tempo máximo de uma comunicação completa no modo DCF (`const MAX_DCF_COM_TIME RTS_TIME + SIFS + CTS_TIME + SIFS + MPDU_MAX_TIME + SIFS + ACK_TIME`).

Esta propriedade foi especificada em $\mathscr{L}_s$, como:

$$\texttt{pc.InitPCF}\texttt{-->}\texttt{pc.Polling\&\&}\ \texttt{pc.maxPCF} <= 2 * \texttt{PIFS} + \texttt{MAX\_DCF\_COM\_TIME} + \texttt{BEACON\_TIME} \tag{6.6}$$

Ao submeter a propriedade para verificação, a ferramenta indicou que esta propriedade se mantém verdadeira para todas as configurações apresentadas. Por não existir participação do PC durante o CP, somente foram verificadas as configurações de PCF exclusivo e de operação conjunta entre os modos.

Propriedade 7 (Consistência nos modos de transmissão)**.**

Quando o modo de acesso é realizado através da PCF as estações DCF não interferem no funcionamento do protocolo, ou seja, uma estação somente transmite no CFP se tiver sido escalonada ou se estiver enviando um ACK para dados previamente recebidos.

Para representar esta propriedade, foi verificado se existe algum caminho computacional onde uma estação qualquer sob a DCF esteja em um estado capaz de iniciar uma transmissão (`CanTransmit`) e o modo de operação seja a PCF. Para representar esta situação, escreveu-se em $\mathscr{L}_s$:

```
E<>(mode==PCF && dcf1.CanTransmit)
```
(6.7)

A propriedade falha, indicando que se o modo atual de operação é a PCF, ou seja, o controle do uso do meio está com o PC, uma estação na DCF não alcançará o estado que indica que ela pode iniciar uma transmissão.

Propriedade 8 (Propriedades de Alcançabilidade)**.**

Estas propriedades podem ser genericamente representadas pela fórmula $\mathscr{L}_s$:

```
E <> Process.State
```
(6.8)

Com `Process` sendo um dos processos compostos paralelamente no sistema, e `State` um dos estados do processo.

Para cada um dos estados de cada um dos processos instanciados, submeteu-se à verificação a correspondente propriedade de alcançabilidade.

As propriedades de alcançabilidade foram utilizadas para verificar se as redes de autômatos fornecidas ao verificador são representativas o suficiente para que todas os estados possíveis de funcionamento do protocolo tenham sido alcançados durante o processo de verificação das propriedades *safety* e *liveness*. Para as maiores configurações, todos os estados do modelo se tornam alcançáveis indicando que os modelos são representativos do comportamento do protocolo.

Tempos de Verificação

Os tempos de verificação em cada uma das propriedades apresentadas, assim como um sumário dos resultados das verificações para as configurações DCF exclusivo, PCF exclusivo e conjunto, são apresentados nas tabelas 6.2 e 6.3.

Tabela 6.2 Tempos de Vericação das Propriedades para diversas configurações

Conf.	Prop. 1 - Ausência de *deadlock*	Prop. 2 - Acesso Exclusivo ao Meio	Prop. 3 - Acesso Exclusivo ao Meio no modo PCF	Prop. 4 - *Livelock* de *backoff*
DCF 1	15ms.	15ms.	15ms.	15ms.
DCF 2	15ms.	15ms.	15ms.	15ms.
DCF 3	35s.328ms.	15ms.	12s.125ms.	15ms.
DCF 4	01h.19m.28s.968ms.	15ms.	21m.18s.906ms	15ms.
PCF 1	15ms.	15ms.	15ms.	***
PCF 2	15ms.	15ms.	15ms.	***
PCF 3	31ms.	15ms.	15ms.	***
PCF 4	4s.515ms.	15ms.	15ms.	***
DCF1 + PCF1	15ms.	15ms.	15ms.	15ms.
DCF2 + PCF2	3m.04s.453ms.	15ms.	01m.03s.421ms.	15ms.
DCF3 + PCF2	02h.00m.07s.843ms.	15ms.	20m.55s.281ms.	15ms.

Conf.	Prop. 5 - *Livelock* de Colisão	Prop. 6 - *Timely Access*	Prop. 7 - Cons. nos modos de transmissão	
DCF 1	15ms.	***	15ms.	
DCF 2	15ms.	***	15ms.	
DCF 3	15ms.	***	10s.890ms.	
DCF 4	19s.656ms.	***	18m.14s.817ms	
PCF 1	***	15ms.	***	
PCF 2	***	15ms.	***	
PCF 3	***	01s.968ms.	***	
PCF 4	***	06s.203ms.	***	
DCF1 + PCF1	15ms.	05s.859ms.	15ms.	
DCF2 + PCF2	13s.578ms.	02m.06s.521ms	01m.02s.171ms.	
DCF3 + PCF2	09m.10s.687ms.	01h.14m.01s.163ms.	19m.02s.327ms.	

Tabela 6.3 Tabela Resumo de Resultado das Verificações

Propriedade	DCF	PCF	Operação Conjunta
1 - Ausência de *deadlock*	Verdadeira	Verdadeira	Verdadeira
2 - Acesso Exclusivo ao Meio	Verdadeira apenas para DCF1	Verdadeira	Falsa
3 - Acesso Exclusivo ao Meio no modo PCF	Verdadeira	Verdadeira	Verdadeira
4 - *Livelock* de *backoff*	Verdadeira apenas para DCF1	***	Falsa
5 - *Livelock* de colisão	Verdadeira apenas para DCF1	***	Falsa
6 - *Timely Access*	***	Verdadeira	Verdadeira
7 - Consistência nos modos de transmissão	Verdadeira	***	Verdadeira

6.4 Análise dos Resultados Obtidos

6.4.1 Propriedades do Protocolo

As propriedades verificadas descrevem as principais características do padrão IEEE 802.11.

A propriedade 1 mostra que o protocolo, em seus modos DCF e PCF, independente se separados ou em conjunto, é livre de *deadlock*. Já as propriedades 2, 4 e 5 indicam que a função de coordenação distribuída fornece ao protocolo uma política de melhor esforço (*best-effort*), dado que a ocorrência de potenciais *livelocks* e colisões não permite o fornecimento de maiores garantias de entrega de mensagens.

Por sua vez, a combinação das propriedades 6 e 7 indicam que o protocolo fornece garantias temporais, mesmo quando ocorre a operação conjunta entre os modos DCF e PCF.

Estas características indicam ser possível a utilização do padrão analisado como canal de comunicação de suporte para protocolos e aplicações com QoS (*Quality of Service*) ou requisitos temporais, desde que estes protocolos/aplicações sejam projetados para executar durante o CFP e segundo a hipótese assumida, ou seja, da não coexistência de BSSs compartilhando as mesmas características da camada física dentro de uma BSA.

6.4.2 Comportamento do Verificador

Com relação à ferramenta de verificação utilizada, o UPPAAL realizou a verificação destas diversas classes de propriedades em tempos aceitáveis para configurações satisfatoriamente complexas. Isto demonstra a adequação da combinação da técnica composicional com a técnica simbólica utilizada pelo algoritmo de verificação de modelos do UPPAAL, para o experimento em questão.

Entretanto, para alcançar a efetiva utilização na indústria, esta ferramenta de verificação ainda carece de algum amadurecimento. O suporte à especificação *top-down*, que para ser utilizada deve ser manualmente controlada pelo especificador e o fornecimento de outras extensões de alto nível (normalmente implementadas em separado pelo especificador) como, por exemplo, a existência de funções aleatórias para representar um grande número de transições entre dois estados, são exemplos desta deficiência, na versão utilizada durante este trabalho.

CAPÍTULO 7

Conclusão

O tempo é o melhor autor. Sempre encontra um final perfeito.

—CHARLES CHAPLIN (Ator e Diretor de Cinema Inglês)

Este trabalho apresenta uma especificação e verificação formal da função de controle de acesso ao meio da subcamada MAC do padrão IEEE 802.11. Nesta especificação foi modelado o comportamento temporal do protocolo e as duas funções de coordenação, PCF e DCF, descritas para o padrão, foram consideradas operando em conjunto.

Para a verificação, procurou-se identificar uma série de propriedades que permitisse aos projetistas de aplicações e sistemas de tempo-real obter um conjunto mínimo de garantias relativas aos canais de comunicação. Dentre tais garantias, pode-se destacar o acesso prioritário ao meio pelo PC, o que garante o início do período livre de contenção em um tempo mínimo finito e determinado, e a manutenção deste período, sem a intervenção de estações que não tenham sido escalonadas, enquanto o PC desejar. Estas propriedades foram formalmente demonstradas e indicam que as redes sem fio IEEE 802.11 são adequadas para a construção de sistemas de tempo-real sem fio.

Este resultado é importante para a construção de sistemas de tempo-real em geral e, em particular, para os sistemas mecatrônicos de tempo-real, tais como os sistemas de controle, os sistemas embarcados e os de robótica colaborativa. Atualmente, estes sistemas demandam cada vez mais suporte à distribuição e à comunicação sem fio, mantendo, entretanto, a necessidade de alto grau de confiabilidade. Vale ressaltar, que é possível se observar o aumento crescente do interesse no uso de redes e padrões comerciais para o projeto de aplicações de tempo-real. Em particular, nota-se o desenvolvimento de diversos trabalhos que consideram as redes IEEE 802.11 como suporte para protocolos e aplicações com requisitos temporais e de QoS.

A falta de implementações comerciais de produtos que suportem a PCF tornam os resultados obtidos ainda mais relevantes. Isto porque este trabalho tornou possível se obter um conhecimento prévio do protocolo e de suas propriedades, antes mesmo do desenvolvimento de protótipos ou da construção em larga escala do *hardware* e *software* que suporte a função de coordenação centralizada.

Adicionalmente, o trabalho apresentou um estudo de caso de uso de um verificador de modelos para a certificação de propriedades de tempo-real para um protocolo sem fio de relativa complexidade. Neste estudo de caso, cenários representativos do protocolo foram utilizados, sem que se tivesse sido exposto ao problema de explosão do espaço de estados. Assim, busca-se com isso consolidar a confiança na aplicação prática do uso de métodos formais para o desenvolvimento de sistemas de tempo-real e sem fio confiáveis.

Para a modelagem do protocolo, partiu-se de uma especificação onde se buscou representar todas as interações entre os processos componentes do sistema (estações, funções de sensoreamento, PC e meio físico) previstas na descrição do protocolo. A partir daí a especificação foi sendo refinada até alcançar um detalhamento tal, que as ações internas, relacionadas ao controle destas interações também fossem representadas. A cada refinamento gerado, as propriedades eram verificadas e, quando necessário, ajustadas. Assim, se tornou possível aumentar o grau de confiança na especificação, em função deste detalhamento progressivo, e se avaliar continuamente se a ferramenta já enfrentava explosão de espaços de estado no nível de detalhamento em questão.

A certificação das propriedades foi realizada em diversas configurações :de uma a quatro estações sob a DCF, de uma a quatro estações sob a PCF, uma estação na DCF e uma na PCF, duas estações na DCF e duas na PCF e três estações na DCF e duas na PCF. Nas maiores configurações utilizadas todos os estados existentes no modelos eram alcançáveis através de algum caminho computacional. Este fato parece indicar serem estas configurações representativas do comportamento geral do protocolo.

Antes deste trabalho, o comportamento temporal do padrão não havia sido satisfatoriamente considerado no que diz respeito à verificação formal do seu comportamento e de suas propriedades. Os trabalhos que tratavam da especificação e/ou verificação do protocolo, não tratavam da operação conjunta das funções de coordenação descritas no padrão, levando em conta apenas uma das funções de operação ou as considerando isoladamente. Além disso, propriedades de tempo não haviam sido suficientemente trabalhadas, pois os trabalhos limitaram-se a considerar a verificação de propriedades de *safety* e *liveness*, sem levar em conta os limites mínimos de tempo para que um determinado estado do protocolo viesse a acontecer (propriedades de *timeliness*). Assim, os resultados obtidos até então não forneciam as garantias necessárias à utilização de redes IEEE 802.11 em sistemas de tempo-real sem fio.

Trabalhos Futuros

Em algumas situações práticas, duas os mais BSAs podem se sobrepor total ou parcialmente. Neste caso, se os *Point Coordinators* compartilharem as mesmas características da

camada física, podem vir a ocorrer colisões de quadros *beacon*, levando a retardos no início do CFP. Modelar estas ocorrências e quais os efeitos impressos nas propriedades verificadas ainda é um objetivo a ser atingido.

Em outras situações, o padrão IEEE 802.11 pode ser utilizado para a construção de sistemas móveis. Nestas situações, o tempo de propagação das mensagens deixa de ser constante e podem acontecer situações de falhas, nas quais as estações podem temporariamente estar inacessíveis umas às outras. Analisar os efeitos da mobilidade sobre as propriedades verificadas é, também, uma questão a ser considerada em trabalhos futuros.

Referências Bibliográficas

ABB+01 Tobias Amnell, Gerd Behrmann, Johan Bengtsson, Pedro R. D'Argenio, Alexandre David, Ansgar Fehnker, Thomas Hune, Bertrand Jeannet, Kim G. Larsen, M. Oliver Möller, Paul Pettersson, Carsten Weise, and Wang Yi. UPPAAL - Now, Next, and Future. In F. Cassez, C. Jard, B. Rozoy, and M. Ryan, editors, *Modelling and Verification of Parallel Processes (MOVEP'2k*, number 2067 in LNCS Tutorial, pages 100–125, 2001.

ACD90 R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real time systems. In *In Proc. 5th Symp. on Logics in Computer Science*, pages 414–425. IEEE Computer Society Press, 1990.

AD90 Rajeev Alur and D. L. Dill. Automata for Modeling Real-Time Systems. In *Proceedings of the Seventeenth International Colloquium on Automata, Languages and Programming*, pages 322–335, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

Ass05 Infrared Data Association. IrDA IrSimple Specifications, 2005.

BBC+99 N. Bjorner, A. Browne, M. Colon, B. Finkbeiner, Z.Manna, B.Sipma, and T.Uribe. Verifying temporal properties of reactive systems: A step tutorial. In *Formal Methods in System Design*, 1999.

BDHS00 Dragan Bosnacki, Dennis Dams, Leszek Holenderski, and Natalia Sidorova. Model checking sdl with spin. In *TACAS '00: Proceedings of the 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 363–377, London, UK, 2000. Springer-Verlag.

BFB01 Beatrice Berard, A. Finkel, and Michel Bidoit. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer-Verlag New York, September 2001.

BGK+96 J. Bengtsson, D. Griffioen, K. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Verification of an audio protocol with bus collision using UPPAAL. In *Proceedings of CAV'96*, 1996. 1102 of Lectures Notes in Computer Science.

BH95 Jonathan P. Bowen and Michael G. Hinchey. Seven more myths of formal methods. *IEEE Computer*, July 1995.

BLL+95 J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL – A tool suite for symbolic and compositional verification of real time systems. In *Proceedings of 1st Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, May 1995. 1019 of Lectures Notes in Computer Science.

Bry86 Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

BS05 Inc. Bluetooth SIG. Bluetooth specification guidelines, 2005.

BY04 Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In W. Reisig and G. Rozenberg, editors, *Lecture Notes on Concurrency and Petri Nets*. Springer–Verlag, 2004. 3098 of Lectures Notes in Computer Science.

CC95 Sérgio Campos and Edmund Clarke. Real-time symbolic model checking for discrete time models. In C. Rattray T. Rus, editor, *Theories and Experiences for Real-Time System Development*, In AMAST Series in Computing, pages 214–223. World Scientific Publishing Company, May 1995.

CDK94 George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley, 2 edition, February 1994.

CGP00 Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, January 2000.

Che02 Albert M. K. Cheng. *Real-Time Systems: Scheduling, Analysis and Verification*. Wiley, John & Sons, July 2002.

CJL02 Wen-Tsuen Chen, Bo-Bin Jian, and Shou-Chih Lo. An adaptive retransmission scheme with QoS support for the IEEE 802.11 MAC enhancement. In *55th IEEE Vehicular Technology Conference*, pages 70– 74, 2002.

Com95 ETSI STC-RES 10 Committee. HIPERLAN Functional Specifications draft Standard - ETS 300-652, 1995.

CR06 Franck Cassez and Olivier H. Roux. Structural translation from Time Petri Nets to Timed Automata. *Journal of Systems and Software*, 2006. forthcoming.

Cro97 Brian P. Crow. IEEE 802.11 Wireless Local Area Networks. *IEEE Communications Magazine*, September 1997.

CV94 T. Chiueh and C. Venkatramani. Supporting real-time traffic on ethernet. In *Proceedings of IEEE Real-time Systems Symposium*, December 1994.

CW96 E. M. Clarke and J. M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, December 1996.

DAC98 Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Property specification patterns for finite-state verification. In *FMSP '98: Proceedings of the second workshop on Formal methods in software practice*, pages 7–15, New York, NY, USA, 1998. ACM Press.

DM01 Alexandre David and M. Oliver Möller. From HUPPAAL to UPPAAL: A translation from hierarchical timed automata to flat timed automata. Research Series RS-01-11, BRICS, Department of Computer Science, University of Aarhus, March 2001.

DMRR00 David Déharbe, Anamaria M. Moreira, Leila Ribeiro, and Vanderlei Moraes Rodrigues. Introdução a métodos formais: Especificação, semântica e verificação de sistemas concorrentes, September 2000.

Dol03 Laurent Doldi. *Validation of Communications Systems with SDL: The Art of SDL Simulation and Reachability Analysis*. John Wiley & Sons, Ltd., June 2003.

FFO00 J. Farines, J. Fraga, and R. Oliveira. *Sistemas de Tempo-Real*. Editora da Universidade Federal de Santa Catarina, July 2000.

FGK96 J. Frößl, J. Gerlach, and Th. Kropf. An efficient algorithm for real-time symbolic model checking. In *European Design and Test Conference*, pages 15–20, March 1996.

Flo67 Robert W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science*, volume 19, pages 19–32. American Mathematical Society, 1967.

GLMR05 Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier (H.) Roux. Romeo: a tool for analyzing time petri nets. In *Proc. 17th International Conference on Computer Aided Verfication (CAV'05)*, volume 3576 of *Lecture Notes in Computer Science*, pages 418–423, Edinburgh, Scotland, UK, July 2005. Springer–Verlag.

Hal90 Anthony Hall. Seven myths of formal methods. *IEEE Computer*, September 1990.

Har87 David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

HHWT97 T. A. Henziger, P. H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. Software Tools for Technology Transfer, 1997. Disponível em http://www-cad.eecs.berkeley.edu/∼tah/HyTech/.

HL97 K. Havelund and K. Lund. Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In *Real-Time Systems Symposium, The 18th IEEE*, pages 2 – 13, 1997.

HLS99 Klaus Havelund, Kim Guldstrand Larsen, and Arne Skou. Formal verification of a power controller using the real-time model checker UPPAAL. pages 277–298, 1999. 1601 of Lectures Notes in Computer Science.

HNSY94 Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems. *Inf. Comput.*, 111(2):193–244, 1994.

Hoa69 C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

Hoa85 C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International Series in Computer Science, April 1985.

HW98 Pao-Ann Hsiung and Farn Wang. A state-graph manipulator tool for real-time system specification and verification. In *Proceedings of the 5^{th} International Conference on Real-Time Computing Systems and Applications - RTCSA'98*, October 1998.

HWG02 Inc. HomeRF Working Group. Homerf specification, 2002.

IEE99 IEEE. ANSI/IEEE Std. 802.11, 1999.

IT00 ITU-T. Specification and Description Language (SDL), 2000.

JB93 M. Eric Johnson and Margaret L. Brandeau. An analytic model for design of a multivehicle automated guided vehicle system. *Manage. Sci.*, 39(12):1477–1489, 1993.

JM86 F. Jahanian and A.K. Mok. Safety analysis of timing properties in real-time systems. In *IEEE Trans. Software Eng.*, volume 12, pages 890–904. IEEE Computer Society Press, 1986.

KK97 R. A. Kemmerer and P. Z. Kolano. Formally specifying and verifying real-time systems. In *Proceedings First IEEE International Conference on Formal Engineering Methods*, pages 112–120, November 1997.

KNP02 M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In J.-P. Katoen and P. Stevens, editors, *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *LNCS*, pages 52–66, Grenoble, April 2002. Springer.

KNS02 M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In H. Hermanns and R. Segala, editors, *Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*, volume 2399 of *LNCS*, pages 169–187. Springer, 2002.

Kop00 Hermann Kopetz. Software engineering for real-time: A roadmap. *Proceedings of the 22nd International Conference on Future of Software Engineering (FoSE) at ICSE 2000, 4th - 11th June 2000, Limerick, Ireland*, June 2000.

KSW96 K. Kravosec, N. Shankar, and P. Ward. Integration of formal verification with real-time design. In *Proceedings of WORDS (Workshop on Object-Oriented Real-Time Dependable Systems)*, pages 128–136, February 1996.

LH02 Jork Loeser and Hermann Hartig. Real time on ethernet using off-the-shelf hardware. In *Proc. of the 1st Intl. Workshop on Real-Time LANs in the Internet Age*, pages 59–62, June 2002.

LL01 Kiu Chang Lee and Suk Lee. Integrated network of Profibus-DP and IEEE 802.11 wireless LAN with hard real-time requirement. In *Proceedings Industrial Eletronics, 2001, Symposium on*, pages 227–234, May 2001.

LLW95 F. Laroussinie, K.G. Larsen, and C. Weise. From timed automata to logic and back. In *MFCS 95*, 1995. Lectures Notes in Computer Science.

LP97 Henrik Lönn and Paul Pettersson. Formal Verification of a TDMA Protocol Startup Mechanism. In *Proc. of the Pacific Rim Int. Symp. on Fault-Tolerant Systems*, pages 235–242, December 1997.

LPY95 Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Model-checking for real-time systems. In *Fundamentals of Computation Theory*, pages 62–88, 1995.

LS01 George Logothetis and Klaus Schneider. Symbolic model checking of real-time systems. In *Temporal Representation and Reasoning TIME*, pages 214–223, June 2001.

McM92 K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, 1992.

McM93 K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, July 1993.

Mil80 Robin Milner. *A Calculus of Communicating Systems*. Lecture Notes in Computer Science Vol. 92. Springer-Verlag, 1980.

MLB+04 Raimundo Macêdo, George Lima, Luciano Barreto, Aline Andrade, Alírio Sá, Frederico Barboza, Rodrigo Albuquerque, and Sandro Andrade. Tratando a previsibilidade em sistemas de tempo-real distribuídos: Especificação, linguagens, middleware e mecanismos básicos. In *XXIIº Simpósio Brasileiro de Redes de Computadores - Livro Texto dos Minicursos*, pages 103–163. Sociedade Brasileira de Computação, May 2004.

MN99 Michael Mock and Edgar Nett. Real-time communication in autonomous robot systems. *ISADS*, page 34, 1999.

MR04 Deborah Estrin William J. Kaiser Mani Srivastava Mohammad Rahimi, Richard Pon. Adaptive sampling for environmental robotics. In *IEEE International Conference on Robotics and Automation*, pages 3537–3544, New Orleans, April 2004.

MS01 Alexandre Mota and Augusto Sampaio. Model-checking CSP-Z: Strategy, tool support and industrial application. *Science of Computer Programming*, (40):59–96, 2001.

Net05 Edgar Nett. Wlan in automation - more than an academic exercise? In Aline Maria S. Andrade Carlos Maziero, João Gabriel Silva and Flávio M. Assis Silva, editors, *Proc. 2nd Latin-American Symposium (LADC 2005)*, volume 3747 of *LNCS*, pages 4–8. Springer, October 2005.

NFA+03 Urbano Nunes, José Alberto Fonseca, Luís Almeida, Rui Araújo, and Rodrigo Maia. Using distributed systems in real-time control of autonomous vehicles. *Robotica*, 21(3):271–281, 2003.

NS03 Edgar Nett and Stefan Schemmer. Reliable real-time communication in cooperative mobile applications. *IEEE Trans. Comput.*, 52(2):166–180, 2003.

Ö03 Peter Csaba Ölveczky. *Real-Time Maude 2.0 Manual*. Department of Computer Science, University of Illinois and Department of Informatics, University of Oslo, October 2003.

Pet62 Carl Adam Petri. *Communication with Automata*. PhD thesis, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962. Second Edition:, New York: Griffiss Air Force Base, Technical Report RADC-TR-65–377, Vol.1, 1966, Pages: Suppl. 1, English translation.

PSvH99 Holger Pfeifer, Detlef Schwier, and Friedrich W. von Henke. Formal verification for time-triggered clock synchronization. In Charles B. Weinstock and John Rushby, editors, *Dependable Computing for Critical Applications—7*, volume 12, pages 207–226, San Jose, CA, 1999. IEEE Computer Society.

PVS03 PVS. The PVS Specification and Verification System, 2003.

Ram99 Krithi Ramamritham. Can real-time systems be built from off-the-shelf components? In *RTCSA*, page 226, 1999.

RK02 J. Ruf and T. Kropf. Formal data analysis of timed finite state systems. In *Real-Time Systems, 2002. Proceedings. 14th Euromicro Conference on*, pages 257–263, June 2002.

Rus05 Mark Russinovich. Process explorer, 2005.

Sch91 S. A. Schneider. An operational semantics for timed CSP. In *Proceedings Chalmers Workshop on Concurrency*, pages 428–456. Report PMG-R63, Chalmers University of Technology and University of Göteborg, 1991.

Sch00 Steve Scheneider. *Concurrent and Real-Time Systems: The CSP Approach*. John Wiley & Sons, Ltd, 2000.

SGZ01 Srikant Sharma, Kartik Gopalan, and Ningning Zhu. Quality of Service guarantee on 802.11 networks. In *Hot Interconnects*, volume 9, pages 99–103, August 2001.

Sha93 Natarajan Shankar. Verification of Real-Time Systems Using PVS. In *CAV '93: Proceedings of the 5th International Conference on Computer Aided Verification*, pages 280–291, London, UK, 1993. Springer-Verlag.

SK97 Devdas Shetty and Richard Kolk. *Mechatronics System Design*. Thomson-Engineering, July 1997.

Smi02 Graeme Smith. An integration of Real-Time Object-Z and CSP for specifying concurrent real-time systems. In *Proceedings of the Third International Conference on Integrated Formal Methods*, pages 267–285. Springer-Verlag, 2002.

Sta88 John A. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, 21, October 1988.

Sta93 John A. Stankovic. Major real-time challenges for mechatronic systems. Technical report, Amherst, MA, USA, 1993.

Sta96 John A. Stankovic. Strategic directions in real-time and embedded systems. *ACM Computing Surveys*, 28(4), December 1996.

SVM00 Andrew P. Snow, Upkar Varshney, and Alisha D. Malloy. Reliability and survivability of wireless and mobile networks. *Computer*, 33(7):49–55, 2000.

Tsu86 T. Tsumura. Survey of automated guided vehicle in a japanese factory. In *IEEE International Conference on Robotics and Automation*, pages 1329– 1334, April 1986.

TV98 E. Tovar and F. Vasques. Real-time fieldbus communications using profibus networks. In *Proceedings of IEEE Transactions on Industrial Electronics*, 1998.

TY99 Y. Tachi and S. Yamane. Real-time symbolic model checking for hard real-time systems. In *Real-Time Computing Systems and Applications, Sixth International Conference on*, pages 496–499, dec 1999.

Wan98 Jiacun Wang. *Timed Petri nets : Theory and application*. Kluwer Academic Publishers, October 1998.

Yov97 S. Yovine. Kronos: A verification tool for real-time systems. Software Tools for Technology Transfer, 1997. Disponível em http://www-verimag.imag.fr/TEMPORISE/kronos/.

YSSC93 Tomohiro Yoneda, Bernd-Holger Schlingloff, Atsufumi Shibayama, and Edmund M. Clarke. Efficient verification of parallel real-time systems. In *CAV - Computer Aided Verification, 5th Conference of*, pages 321 – 346, June 1993. 697 of Lectures Notes in Computer Science.

YVM02 Moustafa A. Youssef, A. Vasan, and Raymond E. Miller. Specification and analysis of the DCF and PCF protocols in the 802.11 standard using systems of communicating machines. In *In Proc. 10th IEEE International Conference on Network Protocols (INCP'02)*, pages 132–141. IEEE Computer Society Press, 2002.

YWB01 Hong Ye, Gregory Walsh, and Linda G. Bushnell. Real-time mixed-traffic wireless networks. *IEEE Transactions on Industrial Electronics*, 48(5), October 2001.

Zha03 Liqiang Zhao. A multi-cell dynamic reservation protocol for multimedia over IEEE 802.11 ad hoc WLAN. In *Proceedings of the 17th International Conference on Advanced Information Networking and Applications (AINA'03)*, 2003.