UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DISSERTAÇÃO DE MESTRADO

ESPECIFICAÇÃO E IMPLEMENTAÇÃO DE COMPONENTES PARA MODELAR REDES LOCAIS SEM FIO AD HOC PADRÃO IEEE 802.11

Leidjane Matos de Souto

Maria Izabel Cavalcanti Cabral
(Orientadora)

Campina Grande – PB
Fevereiro - 2005

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

ESPECIFICAÇÃO E IMPLEMENTAÇÃO DE COMPONENTES PARA MODELAR REDES LOCAIS SEM FIO AD HOC PADRÃO IEEE 802.11

Leidjane Matos de Souto

Maria Izabel Cavalcanti Cabral
(Orientadora)

Dissertação submetida à Coordenação do Curso de Pós-graduação em Informática da Universidade Federal de Campina Grande, como parte dos requisitos necessários para obtenção do grau de mestre em Informática.

Área de concentração: Ciência da Computação.

Linha de pesquisa: Redes de Computadores e Sistemas Distribuídos.

Campina Grande – PB
Fevereiro – 2005

SOUTO, Leidjane Matos de

S726E

Especificação e Implementação de Componentes para Modelar Redes Locais Sem Fio Ad Hoc Padrão IEEE 802.11

Dissertação (Mestrado) - Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande, Pb, Fevereiro de 2005.

98p. Il.

Orientadora: Maria Izabel Cavalcanti Cabral

Palavras Chaves:

1. Simulação Digital
2. Componentes de Software
3. Redes *Ad Hoc* Sem Fio

CDU – 519.876.5

"ESPECIFICAÇÃO E IMPLEMENTAÇÃO DE COMPONENTES PARA MODELAR REDES LOCAIS SEM FIO AD HOC PADRÃO IEEE 802.11"

LEIDJANE MATOS DE SOUTO

DISSERTAÇÃO APROVADA EM 02.02.2005

PROFª MARIA IZABEL CAVALCANTI CABRAL, D.Sc
Orientadora

PROF. ELMAR UWE KURT MELCHER , Dr.
Examinador

PROF. JOBERTO SÉRGIO BARBOSA MARTINS, Dr.
Examinador

CAMPINA GRANDE – PB

“O esforço empregado em uma jornada nunca é pouco quando o objetivo a alcançar é o desejado.”

Leidjane Matos de Souto

Agradecimentos

Primeiramente a Deus, por ele ter me ajudado a vencer todos os obstáculos que surgiram quando da execução deste trabalho.

A todos os professores da COPIN, e especialmente a minha orientadora Maria Izabel Cavalcanti Cabral, que cordialmente me transmitiu os caminhos e as informações necessárias para a execução desta Dissertação.

Aos meus familiares, que em todos os momentos me incentivaram; ao meu noivo Valbério Sales de Medeiros, que me acompanhou em mais essa jornada com todo seu amor, incentivo e apoio.

Aos meus colegas de mestrado pelo convívio.

Às minhas grandes amigas: Fabiana Ferreira, Thiciane Targino e Valéria Cavalcanti - "As Super Poderosas", pela solidariedade, união e companheirismo demonstrados.

Aos funcionários do DSC e da COPIN, em especial a Aninha por sua bondade e gentileza, procurando sempre ajudar de forma muito carinhosa.

Aos alunos de Projeto I e II: Ayran, Cleidson, Marcelino e Adilton com suas contribuições para o enriquecimento deste trabalho. A Flávio Rocha, pela disponibilidade que se prestou a tirar dúvidas e dar sugestões.

Enfim, obrigad a todos que contribuíram em todas as etapas desta Dissertação, para que eu pudesse alcançar vitoriosa, a meta final.

Resumo

Esta Dissertação de Mestrado apresenta a especificação e a implementação de componentes de software visando à construção facilitada de ferramentas de simulação de alto nível para a modelagem e a avaliação de desempenho de redes locais sem fio *ad hoc*, padrão IEEE 802.11. A utilização de uma abordagem de desenvolvimento de software orientada a componentes, explorando a reutilização de software, permite que novas aplicações possam ser construídas visualmente, a partir de um conjunto de componentes interligados, usando um ambiente de desenvolvimento visual. Os componentes apresentados nesta Dissertação representam elementos essenciais para a modelagem de redes *ad hoc*, padrão IEEE 802.11 (fontes de quadros, *hosts* - com a implementação do protocolo de acesso ao meio CSMA/CA, enlace, representando o meio sem fio e sorvedouros, representando a recepção de quadros no destino). Nesta Dissertação, para validar a reutilização dos componentes de redes *ad hoc* mencionada, um ambiente de simulação inserindo esses componentes foi construído. Este ambiente explora também a reutilização de software, englobando outros componentes essenciais a um ambiente de simulação orientado a eventos, tais como: relógio simulado, geradores de variáveis aleatórias, listas de eventos, processador de medidas de desempenho. Como estudo de caso, um simulador de rede *ad hoc* padrão IEEE 802.11 com interface gráfica foi construído. Ressalta-se que os usuários dos componentes apresentados são os desenvolvedores de ferramentas de simulação de rede *ad hoc*, que podem construir simuladores específicos, simplesmente configurando e conectando esses componentes, estendendo suas funcionalidades, ou mesmo, adicionando novos componentes ao ambiente de desenvolvimento.

Abstract

This Master Dissertation presents the specification and the implementation of software components aiming to facilitate the building level high simulation tools that are applied in both modeling and performance evaluation of wireless *ad hoc* networks following IEEE 802.11 standard. The utilization of component-oriented software development approach allows new applications to be constructed visually by means of a set of components linked with each other, using a visual development environment. The software components presented in this Dissertation represent essential elements for the modeling of wireless *ad hoc* networks IEEE 802.11 standard (frame source, hosts - with the implementation of the access protocol CSMA/CA, link and sink). In order to validate the reuse of the components ad hoc networks, a simulation environment inserting those components was built. This environment explores the software reuse including other essential components to the event-driven simulation environment, such as: simulated clock, random value generators, event list, performance measures processor. As a study case, a simulator of wireless *ad hoc* networks IEEE 802.11 standard, with graphic user interface was built. It is pointed out that the users of the presented components are developers of simulation tools for wireless *ad hoc* networks, that can build specific simulators, simply configuring and linking components, extending theirs functionalities, or even, adding new components to the development environment.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

ACK ⇨ Acknowledgement

CSMA/CA ⇨ Carrier Sense Multiple Access/ Collision Avoidance

CTS ⇨ Clear To Send

DFWMAC ⇨ Distributed Foundation Wireless Medium

DIFS ⇨ Distribution Coordenation Function Interframe Space

DSSS ⇨ Direct Sequence Spread Spectrum

FH ⇨ Fixed Host

FHSS ⇨ Frequency Hopping Spread Spectrum

HR-DSSS ⇨ High Rate Direct Sequence Spread Spectrum

IDC ⇨ International Data Corporation

IEEE ⇨ Institute of Electric and Electronic Engineering

ISM ⇨ Industrial Scientific and Medical Band

LAN ⇨ Local Area Network

LLC ⇨ Link Logical Control

MAC ⇨ Medium Access Control

MFC ⇨ Microsoft Fundation Classes

MH ⇨ Mobile Host

MIT ⇨ Massachusetts Institute of Technology

MSS ⇨ Mobility Support Station

NS ⇨ Network Simulation

OFDM ⇨ Orthogonal Frequency Division Multiplexing

OOS ⇨ Object Oriented Simulation

OSI ⇨ Open Systems Interconnection

PARSEC ⇨ Parallel Simulation Environment for Complex Systems

PDA ⇨ Personal Digital Assistants

PIFS ⇨ Point Coordenation Interframe Space

RTS ⇨ Request To Send

SIFS ⇨ Short Interframe Space

WAN⇨ Wide Area Network

WAP ⇨ Wireless Application Protocol

Wi-Fi ⇨ Wireless Fidelity

WLAN ⇨ Wireless Local Area Network

WMAN ⇨ Broadband Wireless Metropolitan Area Network

WPAN ⇨ Wireless Personal Area Network

Capítulo 1

1. Introdução

Neste capítulo contextualiza-se esta Dissertação de Mestrado como um todo. Inicialmente, introduz-se a tecnologia de redes sem fio e as motivações que nos levaram a escolher este tema para Dissertação de Mestrado. Em seguida, são apresentados os objetivos gerais e específicos do trabalho realizado, seu escopo e relevância. Finalmente, apresenta-se a organização desta Dissertação.

1.1. Motivações

As redes sem fio têm, nos últimos anos, ocupado espaço destacável nas tecnologias de transmissão de dados, passando a fazer parte de diferentes ambientes: escritórios, que utilizam microcomputadores, aeroportos, que permitem aos passageiros utilizarem notebooks com acesso à internet, até restaurantes, que utilizam PDAs (*Personal Digital Assistants*) para registrar os pedidos dos clientes.

Padrões e tecnologias de rede sem fio têm surgido, acomodando uma vasta gama de aplicações e coberturas. O destaque para *Wi-Fi* (*Wireless Fidelity* - Fidelidade Sem Fio), nome dado às tecnologias sem fio que respeitam as normas do padrão IEEE 802.11 para as *WLANs* (*Wireless Local Area Networks*), que tem obtido sucesso pela sua grande aplicabilidade, pela sua diversidade em termos de capacidade e cobertura e devido ao baixo custo dos dispositivos de rede [IEEE802.11, a].

Constata-se uma propensão cada vez maior de se implantar as redes sem fio padrão IEEE 802.11 ao invés de redes com fio. Conforme [IEEE802.11, d] essa propensão é motivada pelos seguintes aspectos:

- Inviabilidade da instalação de redes com fio em alguns lugares;
- barateamento dos equipamentos para as redes sem fio;
- interoperabilidade da tecnologia devido à evolução do Padrão IEEE 802.11; e

- velocidade satisfatória que a tecnologia das redes sem fio oferece. Atualmente gira em torno de 11Mbps com o padrão 802.11b, 54Mbps com os padrões 802.11g e 802.11a.

Estas redes permitem uma série de novas funcionalidades para a troca de informações, tais como a facilidade de mobilidade de dispositivos e flexibilidade de conexões. As novas tecnologias de redes prometem o aumento da produtividade com custos relativamente baixos.

Atualmente são definidos, além do padrão IEEE 802.11 para as redes *WLANs,* mais dois padrões de troca de informações sem fios [Tanenbaum, 03]:

- Redes *WPAN* (*Wireless Personal Area Network*) definidas pelo padrão Bluetooth, atualmente incorporado no padrão IEEE 802.15;
- Redes *WMAN* (*Broadband Wireless Metropolitan Area Network*), definidas pelo padrão IEEE 802.16.

Uma outra área de interesse é o acesso à Internet através de dispositivos móveis como celulares e PDAs (*Personal Digital Assistants*). Atualmente são apresentadas duas soluções que permitem esse tipo de acesso: *WAP* (*Wireless Application Protocol*) e i-mode. No entanto, estas tecnologias ainda apresentam limitações quanto à taxa de transmissão e estudos vêm sendo realizados no intuito de aprimorar ou descobrir novas soluções [Loureiro, 03].

Atualmente, os Estados Unidos lideram em uso das tecnologias *Wi-Fi.* Estimativas referentes a 2003 apontavam que 57% das empresas americanas já suportavam as redes 802.11. A adesão ainda mostrava-se maior pelas pequenas empresas e nas casas do que nas grandes corporações, pelas limitações de velocidades (pode chegar a apenas 54Mbps, algo distante dos 100Mbps que se podem alcançar com uma rede *Fast Ethernet* e *Gigabit Ethernet*), alcance (no máximo 100m) e segurança [INFO, 03].

Estatísticas de 2004 mostram que numa lista de 100 empresas de destaque no Brasil, cerca de 45% das empresas de Tecnologia da Informação já usam 802.11b, o protocolo mais popular da tecnologia *Wi-Fi*. Uma pesquisa realizada no Brasil pelo *Yankee Group* com 500 empresas (de 100 funcionários ou mais) apontou que 23% delas aderiram a *WLANs*, 8% pretendiam instalar em seis meses e outros 5% em um ano [INFO, 04].

Contudo, são vários os problemas encontrados nas redes sem fio, considerados grandes desafios a serem solucionados. Entre eles, podem ser citados o gerenciamento da mobilidade dos dispositivos, o baixo desempenho dos protocolos correntes quando utilizados sobre redes

móveis, além de aspectos de segurança. Esses problemas restringem o desenvolvimento e a utilização em larga escala desse tipo de rede.

Devido à complexidade inerente às tecnologias de redes de computadores sem fio demandarem estudos mais detalhados, existe a necessidade de se avaliar esses sistemas de comunicação, visando dar suporte a decisões. Para isso, pode-se usar a técnica da Simulação Digital, que se apresenta como uma importante alternativa para viabilizar esses estudos [Vasconcelos, 02].

A Simulação Digital é uma técnica numérica para execução e construção de um modelo computadorizado de um sistema real ou proposto, com o objetivo de conduzir experimentos numéricos para melhor compreender o comportamento de tais sistemas em condições específicas [Kelton, 98]. Essa técnica apresenta-se como uma importante opção de análise na modelagem e avaliação de desempenho de redes de computadores em geral [Kronbauer, 98].

Com o recente amadurecimento e proliferação das tecnologias para comunicação sem fio, aumentou o interesse por protocolos, serviços e aplicações para redes locais sem fio, conseqüentemente tornou-se imperativo a necessidade de ferramentas de simulação flexíveis que possam facilmente ser adaptadas às necessidades e ao grau de abstração desejado por cada usuário [Mobics2, 03].

A reutilização de software na construção dessas ferramentas apresenta-se relevante e promissora, principalmente, devido à evolução das tecnologias das comunicações que exige, constantemente, novos estudos e análise de seus desempenhos [Rocha, 02].

Entre as formas de reutilização de software, ressalta-se o uso de componentes, que são objetos inteiros já instanciados que podem ser conectados a uma aplicação qualquer [Szyperski, 99]. A composição das aplicações é feita utilizando ferramentas gráficas que permitem configurar e conectar os componentes necessários à aplicação de forma visual.

A construção de ferramentas que possibilitem modelar e avaliar o desempenho de redes locais sem fio *ad hoc* padrão IEEE 802.11 pode ser feita explorando os benefícios do reuso de componentes de software [Vasconcelos, 02]. Um componente representa um pacote coerente de implementação de software que pode ser desenvolvido independentemente e entregue como unidade: tem interfaces explícitas e bem definidas para os serviços que oferece; tem interfaces explícitas e bem definidas para os serviços que requer; e pode ser

composto com outros componentes após a customização de algumas propriedades mas sem modificar os componentes em si [D'Souza, 98].

No entanto, não só código deve e pode ser reutilizado, mas também as próprias fases de análise e de projeto com vistas a reutilizar o conhecimento empregado e as principais decisões tomadas durante estas fases [Freire, 00].

A construção de ferramentas de simulação baseada em componentes requer mudanças no processo de desenvolvimento de modo que, além de desenvolver o produto, deve-se também criar abstrações reutilizáveis [Sauvé, 03]. As metodologias baseadas em componentes focam mais no projeto e codificação, nos quais realmente os componentes e o reuso devem ocorrer, como também assumem a existência prévia de componentes, não tratando o desenvolvimento como uma seqüência de passos nos quais se inicia necessariamente nos requisitos [Garcia, 03].

O Grupo de Redes de Computadores da UFCG tem investido no desenvolvimento de ferramentas de software baseadas em componentes conforme pode ser visto em [Freire, 00], [Wagner, 00], [Lula, 01], [Rocha, 02] e em [Vasconcelos, 02]. Em [Wagner, 00] é apresentado uma especificação de componentes de software que representam as funcionalidades mínimas dos elementos necessários para modelar uma rede TCP/IP. Em [Lula, 01] foi feita uma especificação de novos componentes que representam os elementos essenciais de um ambiente de simulação (escalonador de eventos, relógio, acumuladores estatísticos, entre outros.). Tais especificações, que representaram estudos iniciais, foram realizadas utilizando a linguagem UML (*Unified Modeling Language*), seguindo o processo de desenvolvimento apresentado em [Larman, 98], enfocando as fases planejar e elaborar, análise e o projeto de alto nível (projeto arquitetural).

O trabalho de [Rocha, 02] reutilizou as fases de análise e projeto dos trabalhos de [Wagner, 00] e de [Lula, 01] que foram revistas e estendidas, com base no processo iterativo e incremental descrito em [Larman, 98], a fim de possibilitar a implementação satisfatória dos componentes. Em [Vasconcelos, 02] foi feita uma especificação geral dos componentes necessários à modelagem de redes locais sem fio padrão IEEE 802.11.

O trabalho realizado, descrito nesta Dissertação, encontra-se no âmbito da área de avaliação de desempenho de sistemas discretos, focando na construção de ferramentas de simulação para modelagem e avaliação de desempenho de redes locais sem fio *ad hoc* padrão IEEE 802.11.

1.2. Objetivos da Dissertação

1.2.1. OBJETIVO GERAL

Esse trabalho de Dissertação de Mestrado tem como objetivo principal:

- especificar e implementar componentes de software para serem reutilizados na construção de ferramentas de simulação, específicas para a modelagem e avaliação de desempenho de sistemas de redes locais de computadores sem fio, *ad hoc,* padrão IEEE 802.11. A especificação apresenta recursos do processo de desenvolvimento baseado em componentes (Componentes UML*)* e nesta fase foi reutilizada a especificação de componentes *ad hoc* apresentada em [Vasconcelos, 02]. A tecnologia *JavaBeans* [Sun, 03] foi adotada para a implementação dos componentes. Os componentes, doravante denominados "componentes *ad hoc*", representam os elementos essenciais de modelos de redes sem fio *ad hoc*, quais sejam: Fonte de Tráfego, *Hosts* (origem e destino) - com a implementação do protocolo de acesso ao meio CSMA/CA e Enlace. Para validar a reutilização dos componentes na construção de ferramentas de simulação, um ambiente de desenvolvimento para a construção de ferramentas de simulação inserindo os componentes *ad hoc* foi utilizado, reutilizando o ambiente de simulação apresentado em [Rocha, 02]. Como estudo de caso, um simulador de rede *ad hoc*, padrão IEEE 802.11 (doravante, denominado simulador *ad hoc*) foi construído.

1.2.2. OBJETIVOS ESPECÍFICOS

Os objetivos específicos desse trabalho foram:

- especificar componentes reutilizáveis de software que representam os elementos do modelo de rede *ad hoc* padrão IEEE 802.11, usando recursos do processo de desenvolvimento Componentes UML*,* reutilizando a especificação de componentes *ad hoc* apresentados em [Vasconcelos, 02];
- implementar e testar (teste de unidade) os componentes *ad hoc*; e
- validar a reutilização dos componentes *ad hoc.* Para viabilizar esse objetivo, foram necessários: (i) utilizar um ambiente de desenvolvimento visual para a construção de ferramentas de simulação de redes sem fio*,* reutilizando os componentes *ad hoc* e o ambiente de simulação apresentado em [Rocha, 02] e,

(ii) como estudo de caso, construir um simulador de rede *ad hoc* (teste de sistema dos componentes) e uma interface gráfica para facilitar a submissão de modelos a serem simulados. Para validar esse simulador, modelos de redes *ad hoc* foram simulados e os resultados foram comparados com resultados obtidos através da simulação dos mesmos modelos em outro ambiente de simulação (ambiente *Arena* [Takus, 97]).

1.3. Escopo e Relevância

Os componentes 802.11 *ad hoc* especificados e implementados nesta Dissertação visam a construção facilitada de ferramentas de simulação de alto nível para a modelagem e a avaliação de desempenho de redes locais sem fio *ad hoc,* padrão IEEE 802.11, explorando a reutilização de software. O ambiente de simulação utilizado nessa Dissertação permitiu a construção de um simulador de rede *ad hoc* padrão IEEE 802.11, como estudo de caso, validando a reutilização dos componentes *ad hoc*.

Ressalta-se que os usuários dos componentes ora apresentados são os desenvolvedores de ferramentas de simulação de redes *ad hoc*, que podem construir simuladores próprios, simplesmente configurando e conectando os componentes propostos em um ambiente de desenvolvimento que explora a reusabilidade de software, podendo estender as funcionalidades dos componentes, ou mesmo, adicionar novos componentes. Com o simulador construído, usuários quaisquer, aqui denominados, especialistas em redes, podem montar e submeter seus modelos para simulação.

Existem algumas ferramentas de simulação que possuem diferentes propósitos e características. O *NS* (*Network Simulation*) [NS, 03] apresenta-se como um simulador flexível e permite simulações de protocolos de redes móveis *ad hoc*. Seu código é livre e não faz uso de uma linguagem única, consistindo assim de uma biblioteca de módulos. Essa característica do *NS* gera uma complexidade tanto para o desenvolvedor do ambiente de simulação, que terá que lidar com linguagens distintas, quanto para o analista de modelagem, no sentido de trabalhar em nível de linguagem de *script* para construir os modelos a serem simulados. O *GloMoSim* [GloMoSim, 98] é um simulador de redes sem fio baseado em bibliotecas implementadas em uma linguagem – *Parsec* (*PARallel Simulation Environment for Complex systems*) [Parsec, 98] - que possui limitações, não permitindo a programação de novas abstrações, sendo o seu maior propósito a simulação de redes em larga escala. O *MobiCS* (*Mobile Computing Simulator*) [MobiCS, 03] tem por enfoque a prototipagem, teste e

avaliação de protocolos distribuídos. Existem ambientes de mais alto nível, como é o caso do *BONeS Designer* [Alta Group, 96] e o *OPNET Modeler* [OPNET, 02], entretanto são produtos proprietários.

É essencial que se perceba a diferença dos artefatos de software aqui apresentados com relação às ferramentas de simulação mencionadas. Os componentes apresentados são artefatos de software de alto nível, reutilizáveis, inseridos em um ambiente que permite a construção de simuladores com a tecnologia *ad hoc* IEEE 802.11 de forma rápida e a custo reduzido.

Finalmente, também se destaca como fator relevante, a multidisciplinaridade desta Dissertação, em que são envolvidas três importantes áreas da computação: Simulação Digital, Redes de Computadores e Engenharia de Software (desenvolvimento de componentes de software).

1.4. Organização da Dissertação

Esta Dissertação está organizada em Capítulos e Apêndices. Seguem resumos de seus conteúdos:

O Capítulo 2 apresenta os principais conceitos da técnica da Simulação Digital, mostrando a sua aplicação para a modelagem e avaliação de desempenho de sistemas discretos. São apresentados exemplos de ambientes de simulação voltados para a área de redes de computadores.

O Capítulo 3 introduz os principais conceitos relacionados à tecnologia de redes sem fio, ressaltando particularmente o Padrão 802.11.

No Capítulo 4 é apresentado o processo de desenvolvimento utilizado neste trabalho de dissertação, Componentes UML, juntamente com os artefatos de software que compõem a especificação dos componentes propostos.

O Capítulo 5 apresenta os principais aspectos de implementação e de validação relacionados: (i) aos componentes *ad hoc*; (ii) ao ambiente de desenvolvimento utilizado para a construção de ferramentas de simulação, e (iii) ao simulador de rede, construído como estudo de caso, para fins de validação da reutilização dos componentes *ad hoc*. Ainda neste capítulo, mostra-se a validação desse simulador que consistiu na comparação dos resultados das simulações de modelos obtidos através do simulador *ad hoc* com os resultados obtidos da simulação destes modelos no ambiente *Arena.*

No Capítulo 6 apresentam-se as conclusões do trabalho destacando suas contribuições e sugestões para trabalhos futuros.

Por fim, apresentamos dois Apêndices (A e B). O Apêndice "A" apresenta a tecnologia *JavaBeans* usada para criar componentes de software na linguagem *Java*. Recomendamos a leitura deste Apêndice, antes da leitura dos capítulos 4 e 5, para aqueles que desejam conhecer mais detalhes dessa tecnologia. O Apêndice "B" apresenta todos os componentes de software em detalhes de implementação.

Capítulo 2

2. Simulação Digital

Este capítulo apresenta os principais conceitos da técnica da Simulação Digital, mostrando a sua aplicação para a modelagem e avaliação de desempenho de sistemas discretos. São apresentados exemplos de ambientes de simulação voltados para a área de redes de computadores.

2.1. Introdução

Simular é criar um modelo de um sistema real ou proposto com o objetivo de avaliar o seu comportamento sob várias condições, podendo-se assim tirar conclusões sobre esse sistema sem precisar construí-lo [Kelton, 98].

A modelagem de sistema é muitas vezes preferível para condução de experimentos por permitir conhecer o comportamento e capacidade de sistemas que não estejam implementados como também daqueles já prontos sem prejudicar as atividades do ambiente, sem custar caro e sem levar muito tempo [Silva, 01].

Na avaliação de desempenho deve-se definir todas as informações relevantes para análise do sistema através dos parâmetros a serem medidos, além da forma através do qual a avaliação será feita: se pela técnica da Simulação Digital ou através de Métodos Analíticos.

Os Métodos Analíticos geralmente apresentam solução mais econômica e eficiente, no entanto possui a desvantagem de muitas vezes a sua aplicação ficar limitada ou mesmo inviabilizada devido à complexidade do sistema que está sendo modelado. Nesses casos a Simulação Digital é usualmente a técnica mais indicada.

2.2. Modelagem de Sistemas

Modelo é uma descrição ou abstração do sistema para a resolução de um problema em particular. A sua construção é um processo complexo e considerado uma arte, pois o modelador deve conhecer bem a estrutura e as regras de operação do sistema e saber extrair o essencial do sistema, sem incluir detalhes desnecessários [Soares, 92].

Modelos podem ser classificados de acordo com os métodos utilizados para obtenção dos resultados numéricos. Assim, modelos analíticos são definidos como modelos cuja estrutura é representada por equações matemáticas, através das quais o comportamento do sistema pode ser obtido pela atribuição de valores aos parâmetros do modelo e a solução das equações.

Modelos para simulação podem ser definidos como aqueles representados por uma estrutura matemática/lógica, que pode ser exercitada de forma a mimetizar o comportamento do sistema. Através de experimentos, várias observações são realizadas para dar subsídio às várias conclusões sobre o sistema.

Além da classificação citada anteriormente, os modelos para simulação, segundo [Banks, 96], podem ser classificados com sendo estáticos ou dinâmicos, determinísticos ou estocásticos, e discretos ou contínuos. Um modelo pode ser **estático** ou **dinâmico**, dependendo da importância do fator tempo no modelo. Um modelo estático é aquele no qual as mudanças de estado não envolvem tempo [Almeida, 99]. A maioria dos sistemas em operação ou de interesse de estudos é do tipo dinâmico [Soares, 92]. Exemplos são sistemas de redes de computadores e sistemas de manufatura, onde o tempo decorrido para a mudança de estados é fator relevante.

Modelos de simulação que não contém variáveis randômicas são classificados como **determinísticos**. Modelos determinísticos têm um conjunto de entradas conhecido que resultará em um único conjunto de saídas, ou seja, cada entrada válida tem um efeito preciso e determinado no estado do modelo. Modelos de simulação **estocásticos** têm uma ou mais variáveis randômicas como entradas que levam a saídas randômicas, ou seja, uma entrada válida pode causar várias mudanças no estado do modelo, devido aos processos randômicos.

Um modelo **discreto** é aquele em que as variáveis dependentes variam discretamente em pontos específicos do tempo simulado, referidos como tempo de evento. Modelo de simulação **contínuo** é aquele em que as variáveis dependentes podem variar continuamente ao longo do tempo simulado.

Um modelo não necessariamente considera todos os detalhes do sistema. A quantidade de detalhes incluídos no modelo deve ser baseada no propósito para o qual foi construído. Na construção de modelos, tem-se que estabelecer a estrutura do modelo, determinando os limites do sistema (com o seu ambiente), identificando as entidades, os atributos e as atividades do

sistema, fornecendo valores aos atributos e definindo as interações existentes entre suas entidades.

2.3. O Processo de Simulação

A essência ou propósito da modelagem para simulação é ajudar ao tomador de decisão final na resolução de algum problema. Entretanto, para aprender a ser um bom modelador, deve-se conhecer boas técnicas de solução de problemas juntamente com boas práticas da engenharia de software [Pegden, 95]. Dentro de um processo simulação os seguintes passos de desenvolvimento podem ser identificados:

- Formulação do problema: é a definição clara do problema a resolver e dos objetivos de análise. Consiste da primeira tarefa em um projeto de simulação. É um processo contínuo, pela natureza evolucionária da simulação, portanto, ocorre durante todo o estudo.
- Construção do modelo: após a formulação do problema e definição dos objetivos, o processo de construção do modelo começa. Ao se construir um modelo, define-se os elementos do sistema e suas características, bem como a interação desses elementos causando mudanças no estado do sistema no decorrer do tempo. O modelo deve ser de fácil compreensão, contudo mantendo uma complexidade tal que reflita realisticamente as características importantes do sistema real.
- Determinação dos dados de entrada e saída: a formulação do problema vai gerar os requisitos dos dados de entrada. Os valores de entrada de início são hipotéticos ou baseados em alguma análise preliminar.
- Tradução do modelo: consiste na tradução do modelo para uma forma aceitável pelo computador.
- Verificação: essa fase consiste em determinar se o modelo traduzido executa no computador com esperado. Essa verificação é feita uniformemente através de cálculos manuais.
- Validação: consiste na determinação se o modelo é uma representação razoável do sistema. A validação pode muitas vezes se basear em comparações com outros modelos do sistema já validados, por exemplo, modelos analíticos.
- Plano de tática e estratégia: refere-se ao estabelecimento de condições experimentais para a execução da simulação.

- Experimentação: envolve a exercitação do modelo.
- Análise dos resultados: consiste na interpretação dos resultados.
- Implementação e documentação: por fim, para se considerar completo, tem-se o estágio final de implementação dos resultados e na documentação do modelo para simulação e de seu uso para que seus resultados sejam utilizados.

O processo de simulação sugere que esses passos sejam realizados de forma incremental, adaptadas de acordo com a particularidade do sistema sobre o qual será construído o modelo. Pode-se afirmar que o processo de simulação é realizado através de vários ciclos, onde cada um desses ciclos representa um refinamento do modelo.

2.4. Classificação dos Simuladores

Um sistema pode ser definido como um conjunto de partes organizadas funcionalmente para formar um todo. Um subsistema é uma das partes que permite ser tratada como um sistema isolado. Assim, um sistema é formado por subsistemas [Almeida, 99]. Tal classificação, sistema ou subsistema, irá depender do nível de abstração (detalhamento) que está sendo considerado na análise. Por exemplo, ao considerar a Internet um sistema, um roteador nela pode ser considerado um subsistema. No entanto, o próprio roteador poder ser considerado um sistema e os seus protocolos subsistemas.

Os eventos são atividades que mudam o estado do sistema. As entidades são partes de um sistema sendo vistas como os objetos representados na simulação, descritos pelos atributos e para os quais os eventos ocorrem. Atributos são as propriedades ou características de um sistema ou de uma entidade. O estado do sistema pode ser definido como a descrição de todas as entidades, atributos e atividades num dado ponto do tempo. Por fim, ambiente ou meio de um sistema é o local onde o sistema está inserido. Mudanças no seu ambiente podem afetar o estado do sistema.

Os eventos são geralmente gerenciados pelo uso de listas ou filas [Lula, 01]. Essas listas controlam a execução dos eventos e estes são gerados pelo movimento das entidades pelo sistema. Dessa forma, um evento pode representar a chegada de um cliente em um banco ou início da transmissão de uma mensagem em uma rede de computadores. Um evento é uma perturbação instantânea do sistema ocorrendo em um ponto determinado do tempo.

Os simuladores classificam-se em discretos ou contínuos, ou seja, de acordo com os tipos de modelos que trabalham.

Uma simulação discreta tem por objetivo reproduzir as atividades das entidades engajadas e, a partir daí, conhecer algo sobre o comportamento e desempenho do sistema. O estado do sistema varia discretamente em pontos específicos do tempo simulado, referidos como tempo de evento. O processamento seqüencial dos eventos e a coleta de valores nos tempos de evento fornecem à simulação um comportamento dinâmico [Soares, 92].

Dentro da modelagem para sistemas discretos, os simuladores ainda classificam-se em orientado a evento, ao exame da atividade ou a processo.

Um simulador é orientado a eventos quando o sistema é modelado pela definição das possíveis mudanças de estado que ocorrem no instante de cada evento e a sua execução é realizada pelo processamento da lógica associada a cada evento em uma seqüência ordenada no tempo [Soares, 92]. As mudanças de estado no sistema ocorrem em função dos eventos fazendo com que o estudo destes seja essencial à compreensão do comportamento de um sistema. Dentre outras vantagens, esta abordagem permite conhecer o estado de qualquer entidade do sistema em qualquer instante de tempo.

Em uma simulação orientada ao exame da atividade o modelador descreve as atividades nas quais as entidades do sistema estão engajadas e prescreve as condições que causam o início e o fim de uma atividade.

Um simulador é orientado a processos quando sua execução se dá por meio de uma seqüência de processos em que cada um manipula um conjunto de eventos do mesmo tipo [Wagner, 00]. Nessa abordagem, a seqüência em que os eventos são tratados não é a mesma do sistema real e não há entidades dedicadas ao controle da simulação como acontece com a simulação orientada a eventos.

Na simulação contínua o estado do sistema no modelo é representado por variáveis dependentes que mudam continuamente no tempo. Um modelo para simulação contínua é construído pela definição das equações que definem as relações entre suas variáveis de estado, cujo comportamento dinâmico simula o sistema real.

2.5. Linguagens e Ambientes de Simulação

Com o surgimento de linguagens orientadas à simulação na década de 50, tornou-se mais fácil a modelagem de sistemas. Com o passar dos anos estas linguagens foram se desenvolvendo e outras ferramentas foram adicionadas às linguagens de simulação, de modo a torná-las ferramentas mais poderosas para projeto de sistemas.

Desde a adoção do paradigma de orientação a objetos, várias linguagens têm sido utilizadas para o desenvolvimento e a solução de modelos de simulação. A primeira linguagem orientada a objetos, SIMULA, por exemplo, foi criada objetivando a Simulação Digital. No entanto, outras de propósito geral, não orientadas a objetos, também têm sido utilizadas para tal fim como FORTRAN, Pascal e C. Da mesma forma, linguagens de propósito geral para simulação estruturada (GPSS, SIMScript e SLAM [Kelton, 98]) também foram desenvolvidas. Mais recentemente, surgiram as linguagens de programação de simulação orientada a objetos (linguagens OOS – *Object Oriented Simulation*) [Rocha, 02].

Na simulação de modelos pode-se utilizar linguagens de programação de propósito geral ou específico, ou ambientes de simulação de alto nível.

As linguagens OOS podem ser classificadas de duas formas: (i) de propósito geral como SimJava [Howell, 97] e ModSim-III [ModSim-II, 89], ou (ii) específicas [Roberts, 94] como G2, Taylor ED e Simple++. No primeiro caso, elas podem ser utilizadas para simular os mais diversos tipos de sistemas. No outro caso, são dirigidas a um domínio específico de aplicação.

A vantagem das linguagens e dos ambientes OOS sobre as linguagens e ambientes OO (orientado a objetos) de propósito geral, é que eles possibilitam a redução no tempo de desenvolvimento por possuírem mecanismos e classes específicos voltados à simulação.

Ferramentas de simulação orientadas a objeto (*Object Oriented Simulation – OOS*) têm sido alvo de intensas pesquisas nos últimos anos [Roberts, 94]. A principal razão é o aumento da demanda por software cada vez mais complexo que são por natureza, maiores e mais heterogêneos que outros sistemas, além de na maioria dos casos, serem desenvolvidos e mantidos por várias pessoas. Logo, quanto mais complexo o sistema for, mais benefícios serão obtidos utilizando a OOS [Almeida, 99].

Uma das vantagens de usar a OOS é a possibilidade de modelar sistemas usando entidades presentes no sistema físico. Em um sistema de trânsito por exemplo, pode-se ter pedestres, automóveis e semáforos representados no modelo por essas mesmas entidades. Pensar em termos de objetos permite ao analista uma maior simplicidade e flexibilidade durante a concepção do modelo, possibilitando inclusive, criar subsistemas por meio da herança de comportamento e de atributos e daí interconectá-los a outros subsistemas formando novos sistemas [Almeida, 99].

Ambientes de simulação de alto nível apresentam interfaces amigáveis e permitem a criação e a animação de modelos de sistemas de forma bastante flexível. Seguem exemplos de ambientes de simulação, alguns de domínio público como o *Network Simulator* [NS, 03], *GloMoSim* [GloMoSim, 98], *MobiCS* [MobiCS, 02], *Ptolemy* [Lee, 01] e de caráter comercial como o *BONeS* [AltaGroup, 96] e *Arena* [Takus, 97].

2.5.1. NS

O *NS* (*Network Simulator*) é um simulador de eventos discretos direcionado à pesquisa de redes de computadores [NS, 03]. Ele fornece suporte substancial para simulação de TCP, roteamento, e protocolos multicast em redes com ou sem fio.

O NS sempre recebeu contribuição de diversas instituições de pesquisa tecnológica e de estudantes. A versão 2 do NS foi lançada em 1997, no entanto, ainda não é um produto pronto pois apresenta muitos erros que são aos poucos descobertos e corrigidos muitas vezes pelos próprios usuários.

O NS é um simulador orientado a objetos escrito em C++ que utiliza um interpretador OTcl (*Object Tool Command Language*) como linguagem de modelagem dos cenários. O módulo OTcl , desenvolvido no MIT (*Massachusetts Institute of Technology*), é uma extensão das linguagens Tcl/Tk para a programação orientada a objetos. O simulador suporta uma hierarquia de classes em C++ e uma hierarquia similar no interpretador. Os usuários criam novos objetos no interpretador e tais objetos possuem correspondência de um para um na hierarquia compilada. Devido este modelo de programação, o ns-2 é um simulador de protocolo de propósito geral, flexível, e extensível que não faz uso de uma linguagem única, consistindo de uma biblioteca de módulos.

Entre as principais características do NS destacam-se: classificação de quadros por fluxos, ambiente extensível para a manipulação de filas e escalonamento, suporte estatístico e estrutura extensível de geração de tráfego, desenvolvido para as plataformas Unix (FreeBSD, Linux, SunOS, Solaris) e Windows.

2.5.2. BONES

O BONeS Designer [AltaGroup, 96] é um ambiente de mais alto nível que procura facilitar o processo de modelagem apresentando recursos como animação e interpretação dos

dados gerados, dentre outros. No entanto, esse ambiente é um produto comercializado a preço alto.

2.5.3. GLOMOSIM

O GloMoSim (*Global Mobile Information Systems Simulation Library*) [GloMoSim, 98] é um simulador de redes sem fio baseado em bibliotecas, desenvolvido na UCLA (*University of Califórnia, Los Angeles*). O GloMoSim é composto por uma coleção de módulos de biblioteca implementados em PARSEC (*PARallel Simulation Environment for Complex systems*) [Parsec, 98], uma linguagem baseada em C para descrição de simulações seqüenciais e paralelas. No modelo de programação do PARSEC, os protocolos são implementados em módulos monolíticos chamados *entities* (entidades), permitindo somente a composição vertical de protocolos (protocolos em camadas). As primitivas do PARSEC são insuficientes para a programação de novas abstrações, tais como padrões de mobilidade, de conectividade, de consumo de energia, etc. [MobiCS2, 03]. O enfoque do GloMoSim é ser um ambiente para simulação eficiente de redes em larga escala, enquanto que os objetivos principais dos componentes *ad hoc* são simplicidade de uso e reutilização.

2.5.4. MOBICS

O MobiCS (*Mobile Computing Simulator*) [MobiCS, 02] é um framework de um simulador para a prototipagem, teste e avaliação de protocolos distribuídos para computação. A versão atual do MobiCS contempla apenas redes móveis infra-estruturas, e para estas implementa abstrações como: host móvel (MH – *Mobile Host*), estação de suporte à mobilidade (MSS – *Mobility Support Station*), host fixo (FH – *Fixed Host*), célula, enlaces com fio e sem fio, entre outras.

2.5.5. PTOLEMY

O Ptolemy é um *framework* desenvolvido para modelagem, simulação e projeto de sistemas concorrentes heterogêneos (Lee, 01). O *framework* em sua segunda versão (Ptolemy II) é escrito em Java e possui uma ferramenta visual (Vergil), além de um rico conjunto de componentes e mecanismos de interação. Ele também permite aos usuários uma fácil criação de seus próprios componentes e ambientes de simulação. Os componentes do Ptolemy II podem ser utilizados em projetos avançados e podem interagir com outros componentes desenvolvidos pelos usuários. Todos os programas que fazem parte do *framework* Ptolemy são bem documentados, abertos e gratuitos. Programas mais avançados para aplicações específicas, no entanto, podem ter um custo elevado. Destaca-se aqui a diferença existente

entre o Ptolemy e os componentes *ad hoc*. Enquanto o Ptolemy utiliza o conceito de *framework*, com a utilização dos recursos da orientação a objeto que a linguagem *Java* oferece, para facilitar a extensão de novos componentes, os componentes *ad hoc* além de utilizar também desses recursos, utiliza a tecnologia *JavaBeans*, para facilitar tanto a extensão quanto a reutilização dos componentes *ad hoc* na construção de simuladores de redes *ad hoc* 802.11.

2.5.6. ARENA

O software comercial *Arena* é um ambiente de simulação de propósito geral desenvolvido em 1993, para a plataforma *Microsoft Windows*, pela empresa Norte Americana *Systems Modeling Corporation* [Takus, 97] que é representada no Brasil pela empresa *Paragon Software*.

O ambiente *Arena* foi desenvolvido usando a linguagem Visual *C++* , técnicas da orientação a objetos e recursos da *Microsoft Fundation Classes* (*MFC*). Isto possibilitou a construção de uma *interface* gráfica amigável para o ambiente contendo barras de ferramentas, caixas de diálogo, menus, etc. [Wagner, 00].

O software *Arena* encontra-se disponível em duas versões: Profissional e Acadêmica. A versão acadêmica é gratuita e se diferencia da profissional por não poder utilizar mais de 150 elementos em um modelo, além de não permitir a criação de novos *templates*. A versão profissional apresenta dois ambientes de trabalho [Wagner, 00]: o *Professional Edition* e o *Standard Edition*. O primeiro, permite a construção de novos *templates* pelo usuário. O segundo, é designado à construção, simulação e análise dos modelos do usuário.

É essencial que se perceba a diferença dos artefatos de software apresentados nesta Dissertação com relação às ferramentas de simulação mencionadas. Os componentes *ad hoc* IEEE 802.11*,* são artefatos de software de alto nível, reutilizáveis, inseridos em um ambiente que permite a construção de simuladores de redes com esta tecnologia, de forma rápida e a custo reduzido.

Capítulo 3

3. Redes sem Fio e o Padrão IEEE 802.11

Este capítulo tem por objetivo apresentar os principais conceitos relacionados à tecnologia sem fio, ressaltando particularmente o padrão 802.11. Aspectos relacionados com conceitos físicos serão cobertos. Ênfase maior será dada na parte lógica desses conceitos, como, por exemplo, protocolo de comunicação.

3.1. Introdução

Em uma primeira aproximação, as redes sem fio podem ser divididas em três categorias principais: interconexão de sistemas, LANs (*Local Area Networks*) sem fio e WANs (*Wide Area Networks*) sem fio [Tanenbaum, 03].

Na interconexão de sistemas os componentes de um computador são interconectados usando rádio de alcance limitado, a exemplo da tecnologia Bluetooth. A rede Bluetooth também permite a conexão de câmaras digitais, fones de ouvido, *scanners* e outros dispositivos a um computador apenas trazendo-os para dentro do alcance da rede.

As LANs sem fio representam sistemas em que cada computador tem um modem de rádio e uma antena por meio dos quais pode se comunicar com outros sistemas. Esta categoria está se tornando cada vez mais comum em pequenos escritórios e nas residências, onde a instalação da Ethernet é menos flexível ou tem custo superior. Existe um padrão para LANs sem fio, chamado IEEE 802.11 que está bastante difundido e será detalhado no tópico seguinte.

A terceira categoria de rede sem fio é usada em sistemas geograficamente distribuídos, a exemplo da rede de rádio de baixa largura de banda utilizada para telefonia celular, ou até mesmo as redes de alta largura de banda em que o enfoque é o acesso à Internet de alta velocidade a partir de residências e de empresas comerciais. Também foi definido um padrão para esse serviço, chamado de IEEE 802.16.

Uma outra área de interesse é o acesso à Internet através de dispositivos móveis como celulares e PDAs (*Personal Digital Assistants*). Atualmente são apresentadas duas soluções

que permitem esse tipo de acesso: WAP (*Wireless Application Protocol*) e i-mode. No entanto, estas tecnologias ainda apresentam limitações quanto à taxa de transmissão e estudos vêm sendo realizados no intuito de aprimorar ou descobrir novas soluções [Santana, 04].

3.1.1. HISTÓRICO

A comunicação digital sem fio não é uma idéia nova, desde os primórdios da civilização a vontade da humanidade em comunicação livre de fios fazia-se presente. Na Grécia antiga o uso de sinais de fumaça é mencionado como forma de comunicação.

Várias foram as descobertas que contribuíram para que a comunicação sem fio se tornasse uma realidade. Dentre estas podemos citar [Kelvin, 01]:

- No final do século XVIII, Claude Chape inventa a telegrafia óptica, possibilitando a comunicação sem fio para longas distâncias.
- Em 1820, Hans Christian Oersted descobre experimentalmente que a corrente elétrica produz um campo magnético.
- A contribuição fundamental com Michel Faraday demonstrando a indução eletromagnética em 1831.
- Em 1864, James C. Maxwell lança os fundamentos teóricos sobre campos magnéticos com suas famosas equações.
- Em 1876, Alexander Graham Bell inventa o telefone.
- Finalmente, Heinrich Hertz foi o primeiro a demonstrar, através de um experimento em 1887, as equações de Maxwell sobre ondas eletromagnéticas.

As equações de Maxwell, descrevendo a propagação de ondas eletromagnéticas, e os experimentos de Heinrich Hertz, foram a base para a descoberta da radiotelegrafia por Marconi, o qual conseguiu a primeira patente industrial na área de comunicações sem fio em 1896. Em 1905 ocorreu a primeira transmissão de voz e música em um canal sem fio por Reginald Fessenden.

3.2. Características Físicas do Canal de Comunicação sem Fio

As várias características do canal de comunicação sem fio levam muitas vezes a restrições devido a interferências ou ruídos. Isso torna mais problemático o estabelecimento da conexão entre as unidades de um sistema de comunicação sem fio, diferente de um sistema fixo onde as conexões são todas feitas através de cabos ou fibras ópticas.

As propriedades das ondas de rádio são bastante dependentes da freqüência. Usando baixas freqüências, ondas de rádio podem passar através de obstáculos, enquanto que para freqüências mais altas o sinal está mais susceptível à absorção e a reflexão.

Diferentes atrasos na recepção podem ser causados pela propagação por múltiplos percursos (*multipath propagation*). Esses múltiplos percursos são formados pela reflexão, difração ou espalhamento do sinal transmitido em estruturas próximas ao receptor, tais como edifícios, árvores, postes, morros, etc. A soma dos vários sinais dos múltiplos percursos pode resultar em uma interferência construtiva ou destrutiva do sinal recebido.

O espalhamento temporal dos atrasos devido à propagação por múltiplos percursos caracteriza o *Delay Spread.* Sempre que existe espalhamento temporal pode haver a alteração de amplitude das várias componentes do espectro de freqüências do sinal transmitido. Essa alteração pode ocorrer de maneira uniforme em toda a faixa de freqüências do sinal - configurando o desvanecimento plano - ou poderá afetar somente uma determinada faixa de freqüências configurando o que é conhecido como desvanecimento seletivo.

O movimento relativo entre fonte e receptor ocasiona o efeito *Doppler* que corresponde à percepção de uma freqüência diferente da que está sendo transmitida por uma determinada fonte.

Quando grandes obstáculos, como edifícios, morros e similares se situam entre transmissor e receptor surge o efeito denominado sombreamento (*shadowing*), efeito esse que pode provocar consideráveis quedas na amplitude da potência recebida e interromper momentaneamente a comunicação.

Decorrente de todas estas restrições físicas apresentadas anteriormente, o ambiente de comunicação sem fio apresenta características diferentes em relação a um sistema fixo, como:

- Menor largura de banda
- Freqüentes desconexões (voluntária e involuntária)
- Taxa de erro do canal variável e dependente da localização

3.3. O Padrão IEEE 802.11

Antes um dos maiores problemas de comunicação sem fio era a falta de padronização entre os fabricantes. Antigamente, os produtos de transmissão de dados através de rádio (ou infravermelho) de um fabricante não eram compatíveis com os equipamentos produzidos por

outros fabricantes, mesmo usando a mesma faixa de freqüência para a transmissão dados. A questão não era a faixa de freqüência utilizada, mas sim como os dados eram transmitidos [Torres, 01].

Para solucionar esse problema, o IEEE (*Institute of Electric and Electronic Engineering*) desenvolveu o padrão 802.11, que especifica os níveis físico e enlace para funcionamento de redes sem fio, definindo um protocolo de acesso ao meio para transmissão de dados. Tal protocolo de controle de acesso ao meio é definido como camada de Controle de Acesso ao Meio (MAC). Na Figura 3.1 pode-se ver o modelo de camadas OSI em uma rede usando o padrão 802.11 [Torres, 01].

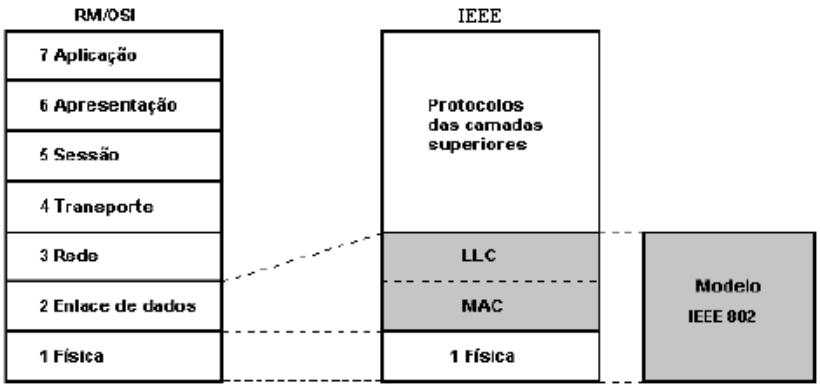

Figura 3.1 - Camadas do padrão OSI com o padrão IEEE 802.11

Entre outras características do projeto IEEE 802.11 pode-se citar como exemplo:

- Suportar diversos canais;
- sobrepor diversas redes na mesma área de canal;
- apresentar robustez com relação à interferência;
- possuir mecanismos para evitar estações perdidas (*hidden node*); e
- oferecer privacidade e controle de acesso ao meio.

Wi-Fi é a abreviatura de "Wireless Fidelity" (Fidelidade sem Fios) e é utilizado para descrever produtos que respeitam o conjunto de normas 802.11 criado pelo *Institute of Electrical and Electronic Engineers* (IEEE).

Com a evolução do padrão IEEE 802.11 e o barateamento dos equipamentos para as redes sem fio, possibilitaram as *WLAN* tornar-se uma realidade. As redes sem fio ficaram mais acessíveis para algumas empresas, aumentando consideravelmente a comercialização de produtos para computadores móveis, com o cartão PCMCIA para Notebook, e o cartão

ISA/PCI para PCs. Na maioria dos casos as empresas utilizam o *Wi-Fi* para interligar redes de dados em vez de recorrer a soluções com cabos. De acordo com a Stanford Conn, 50% das 1000 maiores empresas da revista Fortune vão desenvolver intensamente, até 2005, as suas redes locais sem fios (*WLANs*), recorrendo ao *Wi-Fi* que interliga às redes locais tradicionais (LANs). Em ambientes residenciais as redes locais sem fios (*WLANs*) estão a ter grande adesão para interligar vários PCs a uma ligação externa à Internet.

A evolução do padrão já passou de 11Mbps para taxas de até 54Mbps. O 802.11b utiliza banda de 2,4Ghz e pode transferir dados a uma velocidade de até 11Mbps. O 802.11a trabalha com banda de 5Ghz e consegue transferir dados até 54Mbps. O mais recente standard é o 802.11g que é compatível com o 802.11b e trabalha também com banda de 2,4Ghz e tal como o 802.11a, pode transferir dados até 54Mbps.

3.3.1. ARQUITETURA DO PADRÃO IEEE 802.11

A arquitetura adotada para o padrão IEEE 802.11 para as redes sem fio baseia-se na divisão da área coberta pela rede em células, tais células são chamadas BSA (*Basic Service Area*). Para permitir a construção de redes cobrindo áreas maiores que uma célula, múltiplas BSAs são interligadas através de um sistema de distribuição via pontos de acesso (*Access Point* – Aps). Os Aps são estações especiais responsáveis pela transmissão e captura de dados realizados pelas estações de sua BSA, destinadas a estações localizadas em outras BSAs. Vários BSAs interligados por um sistema de distribuição através de Aps definem uma ESA (*Extended Service Area*).“O conjunto de estações formado pela união dos vários BBSs (*Basic Service Set*) conectados por um sistema de distribuição define um ESS (*Extended Service Set*)”[Torres, 01].

Um ESS é formado pela interconexão de vários BBSs constituindo uma rede sem fio com infra-estrutura. A infra-estrutura é formada por pontos de acessos que interligam o sistema. O sistema de distribuição além de interligado por vários Aps pode fornecer recursos necessários para interligar a rede sem fio a outras redes. Na Figura 3.2 observa-se o sistema de redes sem fio com infra-estrutura.

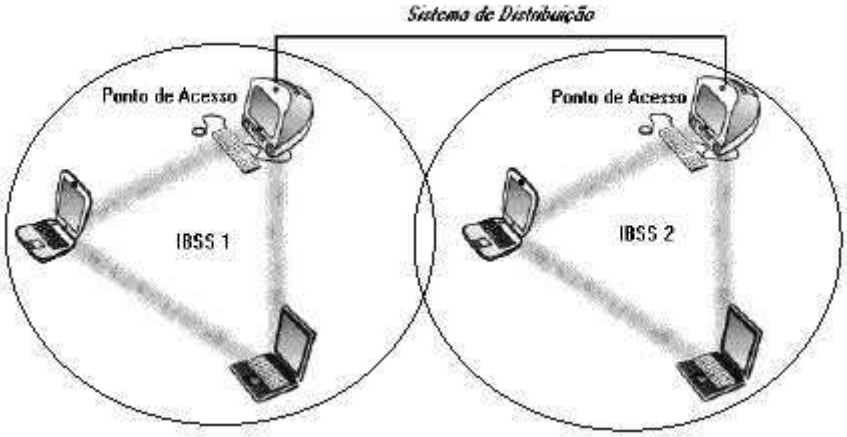

Figura 3.2 – Rede Infra-Estruturada [Vasconcelos, 02]

As estações que funcionam como pontos de acesso têm as seguintes características com relação ao sistema de distribuição:

- Autenticação: permite que estações continuem conectadas à infra-estrutura, mesmo que estações por ventura mudem de uma BSA para outra.
- Associação: estações utilizam um sistema de procura de sinal, escolhendo o melhor ponto de acesso.
- Reassociação: com a associação estabelecida ao novo ponto de acesso, a estação passa a acessar o sistema de distribuição através do AP escolhido.
- Gerenciamento de Potência: permite que as estações operem economizando energia, para isso o AP armazena temporariamente quadros endereçados a estações que estão poupando energia. Periodicamente as estações ligam seus receptores e o AP transmite quadros anunciando tráfego, para que as estações possam se preparar para receber os quadros a elas endereçadas que estão armazenados no AP.
- Sincronização: esta função deve garantir o sincronismo das estações associadas a um AP, por um relógio comum.

Um tipo especial de rede sem fio, nessa arquitetura, é quando se tem uma rede em que o ESS é formado por um único BSS. Tal tipo de rede é denominado rede local sem fio *Ad Hoc*, mostrada na Figura 3.3. Uma rede *ad hoc* permite a comunicação entre estações que estejam próximas umas das outras. Essa arquitetura não utiliza infra-estrutura.

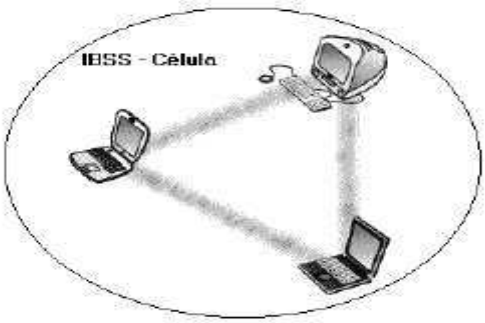

Figura 3.3 - Rede *Ad Hoc* [Vasconcelos, 02]

3.3.2. NÍVEL FÍSICO DO 802.11

As redes sem fio, assim como outras tecnologias de comunicação, utilizam o espaço como meio físico de transmissão. Para que não haja interferência entre essas tecnologias são estabelecidas diversas faixas de transmissão. O padrão 802.11 de 1997 especifica três técnicas de transmissão permitidas na camada física: infravermelho, FHSS (*Frequency Hopping Spread Spectrum*) e DSSS (*Direct Sequence Spread Spectrum)*. Todas elas operam a 1 ou 2 Mbps e as duas últimas utilizam a banda ISM (*Industrial Scientific and Medical Band*) de 2,4 GHz [Tanenbaum, 03]. Em 1999, com o objetivo de se alcançar uma maior largura de banda, foram apresentadas duas novas técnicas: OFDM (*Orthogonal Frequency Division Multiplexing*) e HR-DSSS (*Rate Direct Sequence Spread Spectrum*), que operam em até 54Mbps e 11Mbps, respectivamente. A Tabela 3.1 ilustra um resumo dos padrões 802.11 quanto ao tipo de transmissão na camada física.

802.11a	802.11b	802.11g
5 GHz, 54 Mbps	2,4 GHz, 11 Mbps	2,4 GHz, 54 Mbps
OFDM	DSSS, FHSS	OFDM
não compatível com a, b	base instalada	upgrade da 802.11b
maior capacidade	menor capacidade	capacidade moderada
interferência limitada	interferência potencial	interferência potencial
padronização não mundialmente aceita	padronização mundialmente aceita	padronização mundialmente aceita

Tabela 3.1 – Padrões x Técnicas de Transmissão

3.3.2.1. Infravermelho

Existem, basicamente, duas modalidades de enlace curto por infravermelho: a conexão por linha de visada, na qual dois pontos são interconectados por um feixe diretivo que vai do transmissor ao receptor; e o chamado infravermelho difuso, no qual o transmissor "inunda" o recinto com luz infravermelha que é lançada em todas as direções, reverberando pelas paredes e assim alcançando todos os receptores do recinto.

No infravermelho por linha visada, a ausência de múltiplos caminhos entre transmissor e receptor permite que se alcancem taxas maiores de transmissão, mas o sistema é totalmente vulnerável à presença de obstáculos que eventualmente se coloquem na linha de visada. Em redes Ethernet, as taxas podem alcançar até 10 Mbps, e em redes do tipo token ring até 16 Mbps.

Para aplicações ponto-multiponto ou difusão, é necessário usar o infravermelho difuso, mas aí as taxas caem devido à propagação por múltiplos caminhos. Uma maneira interessante de amenizar este problema é a transmissão quase-difusa, na qual todos os transmissores apontam para um refletor (que pode ser passivo ou ativo) situado no teto do recinto numa posição central. A reverberação fica então limitada à superfície do refletor, reduzindo a variação de comprimento dos percursos.

O padrão 802.11, quando através de infravermelho, utiliza a transmissão difusa, sendo permitidas duas velocidades: 1Mbps ou 2Mbps. Como os sinais não podem atravessar paredes, células situadas em salas diferentes ficam bem isoladas umas das outras. No entanto, devido à baixa largura de banda, e ao fato de que a luz solar altera os sinais de infravermelho, essa não tem sido uma opção popular [Tanenbaum, 03].

3.3.2.2. DSSS

O método de comunicação por espalhamento espectral foi desenvolvido durante a Segunda Guerra Mundial, de forma a tornar a comunicação dentro de um campo de batalha confiável e segura [Santana, 04]. O DSSS (*Direct Sequence Spread Spectrum*) foi o primeiro método de comunicação por espalhamento espectral desenvolvido e envolve o espalhamento do sinal de comunicação, de banda estreita, numa banda de comunicação mais larga através da multiplicação do mesmo por uma seqüência pseudo-aleatória produzida por gerador de códigos pseudo-aleatórios (Figura 3.4). Para que a informação seja recebida pelo receptor, o mesmo necessita utilizar o mesmo código em perfeito sincronismo.

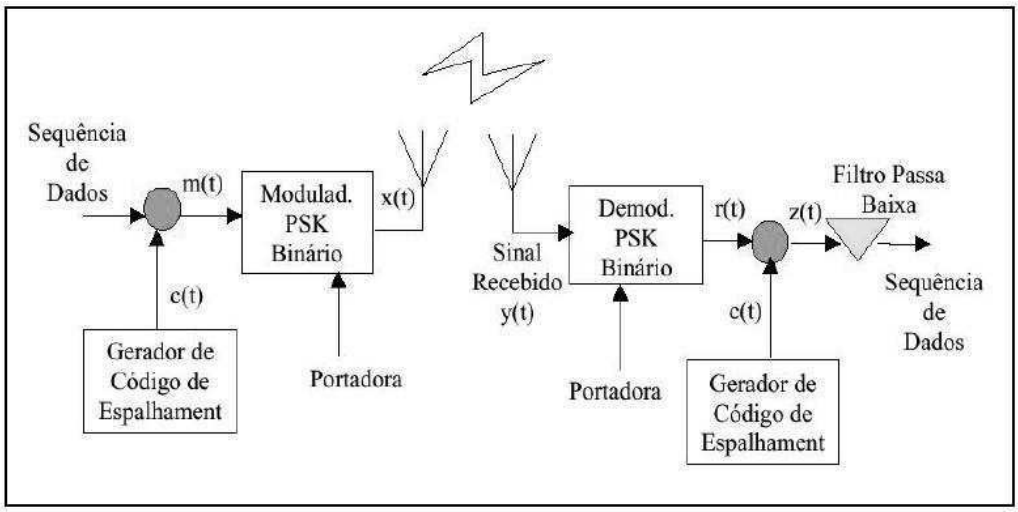

Figura 3.4 – Diagrama de Blocos do DSSS

Como o sinal fica espalhado por uma larga faixa de freqüência caso ocorra alguma interferência em uma determinada freqüência, esta afetará apenas uma pequena parte do sinal. Além disso, a detecção através de uma simples análise espectral se torna complicada, uma vez que devido à baixa densidade do sinal, este fica comparável e conseqüentemente confundível com um ruído térmico. Apesar dessas vantagens no que se refere à segurança o DSSS limita-se à taxa de transmissão de apenas 1Mbps ou 2Mbps.

3.3.2.3. FHSS

O FHSS (*Frequency Hopping Spread Spectrum*) é a mudança periódica de freqüências dentro de uma banda escolhida, seguindo uma seqüência determinada através de um código pseudo-aleatório [Santana, 04]. Utiliza 79 canais, cada um com 1MHz de largura, começando na extremidade baixa da banda ISM de 2,4GHz. As estações devem estar sincronizadas e permanecem transmitindo naquela determinada freqüência por um período de tempo pré-estabelecido (tempo de parada) e que não deve ser superior a 400ms [Tanenbaum, 03]. Isso impede que um intruso que não conheça a seqüência de saltos nem o tempo de parada possa espionar uma transmissão – Figura 3.5.

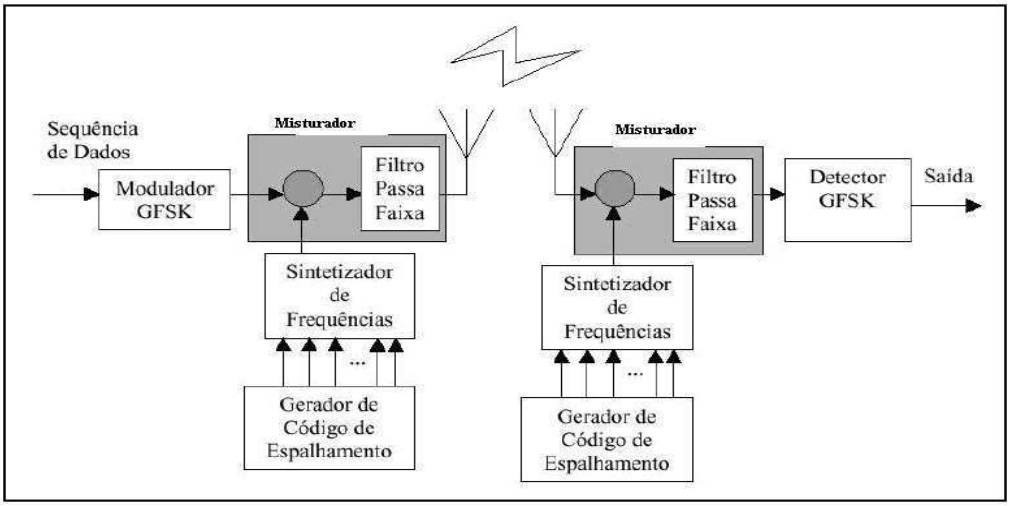

Figura 3.5 – Diagrama de Blocos do FHSS

Os padrões 802.11 e Bluetooth, embora utilizem técnicas de espalhamento espectral, não fornecem privacidade na camada física, pois utilizam código e seqüência conhecidos.

O FHSS apresenta menor consumo de potência em relação ao DSSS, sendo mais utilizado por aparelhos portáteis que dependem de bateria.

3.3.2.4. OFDM

A primeira das LANs sem fio de alta velocidade, a 802.11a, utiliza OFDM (*Orthogonal Frequency Division Multiplexing*) para transmitir até 54Mbps na banda ISM mais larga, de 5GHz. São usadas 52 freqüências, sendo 48 para dados e 4 para sincronização. A divisão e transmissão simultânea do sinal em várias bandas estreitas têm algumas vantagens, como por exemplo, melhor imunidade a interferência de banda estreita e a possibilidade de usar bandas não-contíguas.

3.3.2.5. HR-DSSS

O HR-DSSS (*High Rate Direct Sequence Spread Spectrum*) utiliza 11 milhões de chips/s para alcançar 11Mbps na banda de 2,4GHz, e é adotado pelo 802.11b. As taxas de dados admitidas por este padrão são 1, 2, 5,5 e 11 Mbps. As duas taxas mais baixas funcionam a 1 Mbaud, com 1 e 2 bits por baud, respectivamente, usando a modulação por

deslocamento de fase. As duas taxas mais rápidas funcionam a 1,375 Mbaud, com 4 e 8 bits por baud, respectivamente, usando códigos Walsh/Hadamard [Tanenbaum, 03].

Embora o 802.11b seja mais lento que o 802.11a, seu alcance é cerca de sete vezes maior, o que é mais importante em muitas situações.

3.3.3. NÍVEL DE ENLACE DO 802.11

O nível de enlace é responsável por algumas das funções mais importantes de comunicação numa rede local de computadores. Seu papel é oferecer uma interface de serviços de comunicação para os níveis superiores da arquitetura (aplicações), transformando as informações em unidades chamadas quadros, e fazendo o acesso ao meio físico para a efetiva transmissão desses quadros [Tanenbaum, 03].

Na arquitetura IEEE 802.X o nível de enlace é subdividido em duas subcamadas: subcamada controle de enlace lógico (*Link Logical Control* – LLC), que é definido pelo padrão 802.2 para todas as tecnologias da família IEEE 802, e a subcamada de acesso ao meio (*Medium Access Control* – MAC), que corresponde ao padrão 802.11, no caso das redes locais de computadores sem fio [IEEE802.11, d]. A subcamada MAC implementa o protocolo de acesso ao meio, usado para transmissão de dados.

3.3.3.1. Protocolo MAC 802.11

Além de definir um mecanismo para transmissão física usando radiofreqüência ou infravermelho, o IEEE definiu um mecanismo de acesso ao meio (subcamada MAC do nível de enlace de dados), denominado de DFWMAC (*Distributed Foundation Wireless Medium Access Control*), que suporta dois métodos de acesso: um método distribuído básico, que é obrigatório; e um método centralizado, que é opcional, podendo esses dois métodos coexistir [IEEE802.11, d]. O protocolo de acesso ao meio das redes 802.11 também trata de problemas relacionados com estações que se deslocam para outras células (*roaming*) e com estações perdida (*hidden node*).

O método de acesso distribuído forma a base sobre a qual é construído o método centralizado. Os dois métodos, que também podem ser chamados de funções de coordenação (*coordination functions*), são usados para dar suporte à transmissão de tráfego assíncrono ou tráfego com retardo limitado (*time bounded*).

Uma função de coordenação é usada para decidir quando uma estação tem permissão para transmitir. Na função de coordenação distribuída (*Distributed Coordination Functions* -

DCF), essa decisão é realizada individualmente pelos pontos da rede, podendo, dessa forma, ocorrer colisões. Na função de coordenação centralizada, também chamada de função pontual (*Point Coordination Function* - PCF), a decisão de quando transmitir é centralizada em um ponto especial, que determina qual estação deve transmitir em que momento, evitando teoricamente a ocorrência de colisões [Soares, 95]. Seguem detalhes do funcionamento dessas duas funções.

Função de Coordenação Distribuída (DFC)

O modo DCF, também conhecido como acesso com contenção, utiliza um protocolo chamado CSMA/CA (*Carrier Sense Multiple Access/Collision Avoidance*). O CSMA/CA tem como objetivo evitar que ocorram colisões, uma vez que estas são muito difíceis de detectar em um ambiente sem fio. Para isso, ele funciona da seguinte maneira:

- Se o meio estiver livre por um período (DIFS – *Distribution Coordination Function Interframe Space*) maior que 50μs, então a estação pode transmitir.
- Se o meio estiver ocupado, a estação executará o procedimento de *backoff*, ou seja, esperar que o meio seja desocupado. Este tempo é composto de um tempo mínimo igual ao DIFS somado a outro tempo determinado aleatoriamente.
- Passado esse período de contenção, as estações vão subtraindo o tempo de contenção do tempo total de espera, e a estação que tiver o menor valor terá o direito de utilizar o meio.
- As estações que não transmitiram devem esperar até a primeira estação terminar de transmitir e só então poderão tentar transmitir novamente. Neste caso, o tempo de espera não é um novo tempo aleatório e sim um tempo residual, ou seja, o tempo restante da tentativa anterior.

No método CSMA/CA podem ocorrer colisões e esse método não garante a entrega correta dos dados. Com isso, uma estação após transmitir um quadro, necessita de um aviso de recebimento que deve ser enviado pela estação destino. Para isso, a estação que enviou o quadro aguarda um tempo (*timeout*) pelo aviso de recebimento do quadro por parte da estação destino. Caso esse aviso não chegue no tempo considerado, a estação origem realiza novamente a transmissão do quadro.

Para melhorar a transmissão de dados, o mecanismo de acesso DFWMAC acrescenta ao método CSMA/CA com reconhecimento, um mecanismo opcional que envolve a troca de

quadros de controle RTS (*Request To Send*) e CTS (*Clear To Send*) antes da transmissão de quadros de dados. Esse mecanismo funciona da seguinte forma [Soares, 95]:

- Uma estação antes de efetivamente transmitir o quadro de dados, transmite um quadro de controle RTS, que carrega uma estimativa da duração no tempo da futura transmissão do quadro de dados.
- A estação de destino, em reposta ao quadro de controle RTS, envia um quadro de controle CTS avisando que está pronta para receber o quadro de dados. Só então, a estação transmissora envia o quadro de dados, que deve ser respondido com um reconhecimento (*ack*) enviado pela estação receptora.

O quadro RTS basicamente possui as funcionalidades de reservar o meio para a transmissão do quadro de dados, e de verificar se a estação de destino está pronta para receber o quadro de dados, sendo esta última funcionalidade devido à possibilidade da estação de destino estar operando no modo de economia de energia (modo "*power save*").

O mecanismo básico do controle de acesso DFWMAC é ilustrado na Figura 3.6. Nela pode-se observar que uma estação com quadros para transmitir deve inicialmente sentir o meio livre por um período de silêncio mínimo, IFS (*Inter Frame Space*), utilizando valores diferentes para esse período.

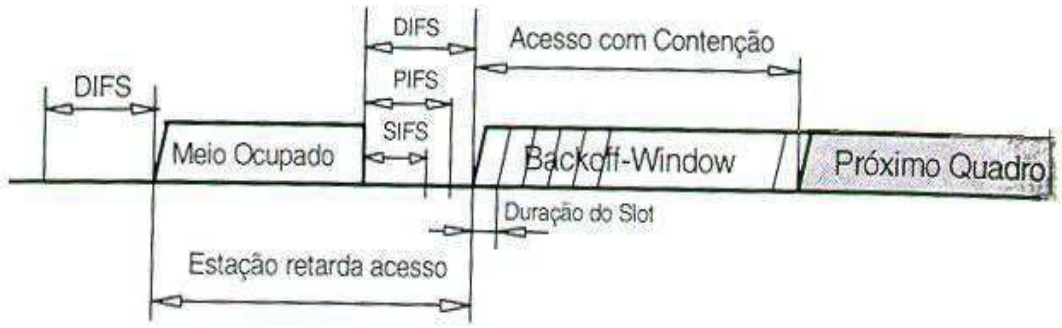

Figura 3.6 – Método de acesso CSMA/CA

O DFWMAC define três prioridades de acesso ao meio [Soares, 95]:

- ***Distributed InterFrame Space* (DIFS)** – espaçamento entre quadros da DFC (Função de Coordenação Distribuída), este parâmetro indica o maior tempo de espera. Ele monitora o meio, aguardando no mínimo um intervalo de silêncio para transmitir os dados.

- ***Point Coordenation InterFrame Space*** **(PIFS)** – espaçamento entre quadros da PFC (Função de Coordenação Pontual), um tempo de espera entre o DIFS e o SIFS (prioridade média) envia quadros de contenção de superquadros, é usado para o serviço de acesso com retardo.
- ***Short InterFrame Space*** **(SIFS)** – é usado para transmissão de quadros carregando respostas imediatas (curtas), como ACKs.

Função de Coordenação Pontual (PCF)

Trata-se de uma função opcional que pode ser inserida ao mecanismo de acesso DFWMAC, sendo construída sobre uma função de coordenação distribuída (DCF) para transmissões de quadros assíncronos, e é implementada através de um mecanismo de acesso ordenado ao meio, que suporta a transmissão de tráfego com retardo limitado ou tráfego assíncrono [Soares, 95].

Para a integração dessas duas funções – pontual e distribuída – é utilizado o conceito de superquadro. Cada superquadro compreende um período livre de contenção (modo PCF) e um período com contenção (modo DFC), como mostra a Figura 3.7. Durante os períodos nos quais as estações estão no modo PCF, o coordenador de ponto consulta se cada estação tem algo a transmitir. As estações recebem dados quando são consultados pelo coordenador de ponto.

O coordenador de ponto inicia e controla o tempo livre de contenção. Ele escuta o meio por PIFS (*Point Coordination Interframe Space*) segundos e então começa um período livre de contenção (*Contetion Free Period* - CFP) através da difusão de um sinal de "*beacon*". Como, por definição, PIFS é menor que DIFS, nenhuma estação pode começar a enviar dados no modo DCF antes do coordenador de ponto. Todas as estações adicionam a duração máxima do período de contenção (*CFPmaxduration*) aos seus respectivos NAVs. O período livre de contenção pode terminar a qualquer momento através do envio de um quadro Cfend pelo coordenador do ponto. Isso ocorre freqüentemente quando a rede está com pouca carga. Além disso, o início de um período livre de contenção pode ser adiado por causa da transmissão de alguma estação no modo DCF (atraso da Figura 3.7).

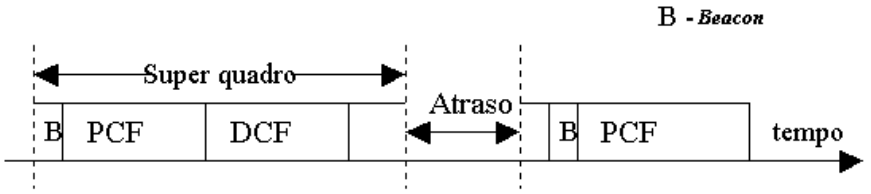

Figura 3.7 – Função de Coordenação Pontual

Quando chega a vez de uma estação transmitir, o coordenador de ponto envia um quadro de dados, caso exista algum a ser enviado dentro de um quadro de consulta (*piggyback*). O receptor envia de volta um ACK, também com dados, se for o caso, depois de SIFS segundos. Após encerrar a transmissão a todas as estações contidas em uma lista de consultas, o coordenador de ponto reinicia o processo de consulta após PIFs segundos. Os usuários que estão sem transmitir por alguns ciclos são retirados da lista de consultas e são consultados de novo no início do próximo período livre de contenção.

Capítulo 4

4. Especificação de Componentes para Modelar Redes *Ad Hoc* Sem Fio Padrão IEEE 802.11

Este capítulo tem por objetivo apresentar o processo de desenvolvimento utilizado neste trabalho de dissertação Componentes UML, juntamente com os artefatos que compõem a especificação dos componentes ora propostos. Essa especificação teve como ponto de partida os trabalhos realizados em [Rocha, 02] e em [Vasconcelos, 02].

4.1. O Processo de Desenvolvimento

São várias as metodologias de desenvolvimento de componentes conhecidas e utilizadas tais como Componentes UML [Cheesman, 01], KobrA [Atkinson, 01] e Calalysis [D'Souza, 98]. A escolha da metodologia de desenvolvimento Componentes UML deveu-se ao fato da observação dos aspectos de simplicidade e facilidade de aplicação desta. Uma vez que este trabalho de dissertação é multidisciplinar, abrangendo as áreas de Redes de Computadores, Engenharia de Software e Simulação Digital, alguns artefatos ou atividades não indicados nessa metodologia, tais como, diagrama de estados e especificação dos requisitos em funcionais e não-funcionais, foram elaborados e adicionados à metodologia adotada.

A metodologia para Componentes UML consiste das seguintes etapas:

- Definição de Requisitos: através dos artefatos desta etapa, objetiva-se tornar mais claro o escopo almejado antes de começar a construção efetiva dos componentes. Para isso, são construídos os artefatos correspondentes ao Modelo de Processo, Modelo Conceitual e Modelo de Casos de Uso.
- Modelagem de Componentes: esta etapa tem por objetivo gerar um conjunto de especificações de componentes e interfaces que serão reunidos em seguida para originar a aplicação referente ao domínio do problema. Essa etapa é subdividida em Identificação dos Componentes, Identificação das Interações entre os Componentes e Especificação dos Componentes.

- Materialização de Componentes: é nesta etapa em que os componentes especificados são implementados. Essa implementação pode ser feita diretamente (construindo o componente) ou através do reuso de componentes que satisfaçam a especificação. Nessa etapa é preciso se preocupar com o ambiente em que o componente será executado, pois o ambiente estabelece um conjunto de regras que devem ser obedecidas e um conjunto de serviços de infra-estrutura do qual o componente possa necessitar.
- Montagem da Aplicação: consiste da etapa final do desenvolvimento de software baseado em componentes em que os componentes previamente implementados ou reusados e testados individualmente, são reunidos em um sistema. Uma interface de usuário para este sistema é projetada de forma a obter uma aplicação.
- Testes: ao final da etapa de montagem, a aplicação passa pela etapa na qual são realizados os testes de aceitação do usuário.
- Distribuição da Aplicação: após a realização dos testes, a aplicação estará pronta para ser entregue aos seus usuários.

A Figura 4.1 fornece uma visão geral das etapas do processo.

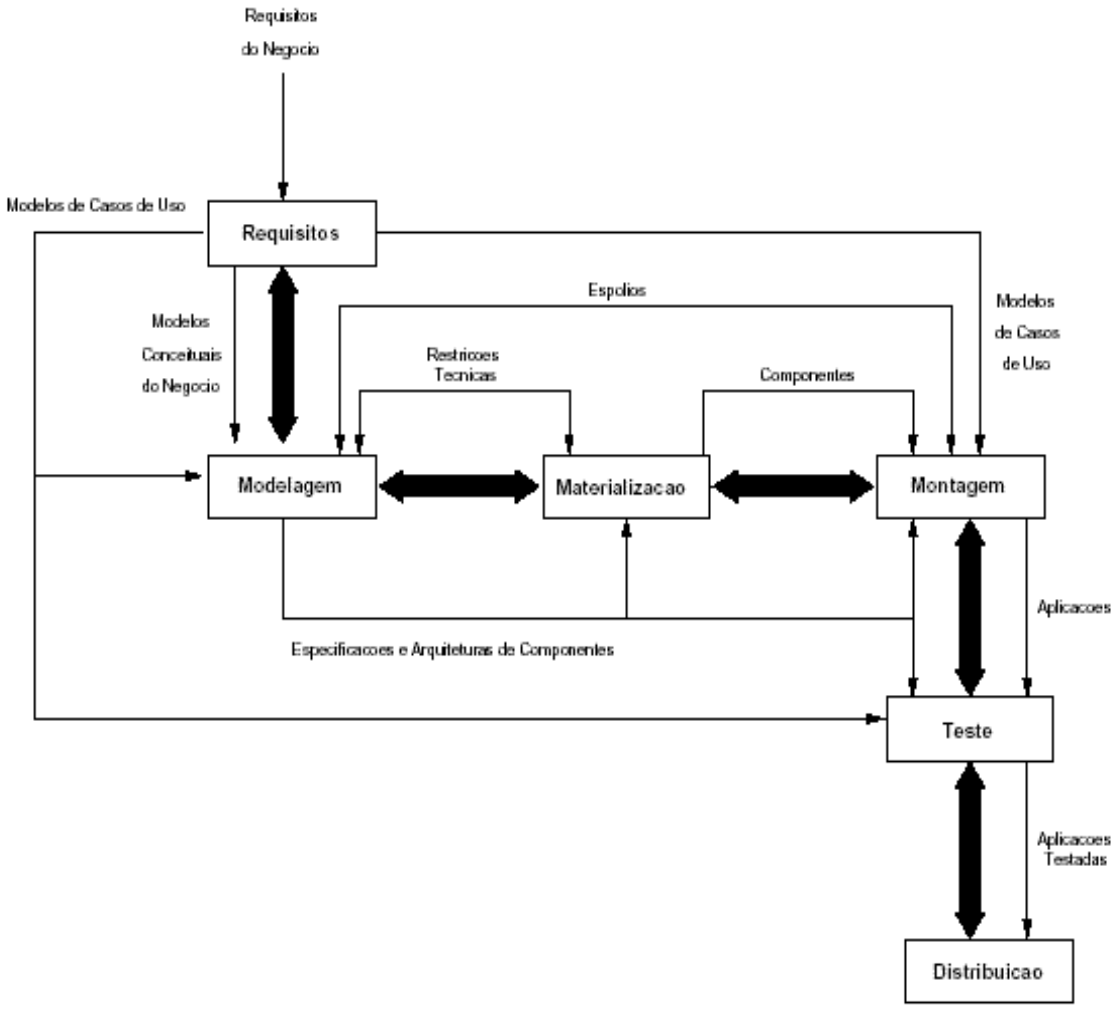

Figura 4.1 – Etapas do Processo de Desenvolvimento Componentes UML

4.2. Definição dos Requisitos

O objetivo desta especificação é representar os componentes essenciais que podem ser usados na construção de ferramentas de simulação de alto nível para modelagem e avaliação de desempenho de redes locais sem fio *ad hoc* padrão 802.11. Para isso, é preciso conhecer o contexto que define o escopo do problema, destacando as restrições e suposições feitas para viabilizar o objetivo principal, que é a especificação e implementação do protocolo de acesso ao meio da subcamada MAC (CSMA/CA com reconhecimento).

Para redes sem fio IEEE 802.11 são especificados dois modos de acesso ao meio físico: modo de acesso sem contenção e com contenção. O modo de acesso sem contenção, também chamado de PCF (*Point Coordination Function* – Função de Coordenação Pontual), ocorre somente no modo infra-estruturado, na presença de um Ponto de Acesso. Neste modo, o Ponto de Acesso pergunta a cada uma das estações associadas a ela se desejam transmitir. O

modo de acesso com contenção, também chamado de DCF (*Distributed Coordination Function* – Função de Coordenação Distribuída), utiliza o protocolo CSMA/CA (*Carrier Sense Multiple Access/Collision Avoidance*). O CSMA/CA tem como objetivo evitar que aconteçam colisões, já que as mesmas são muito difíceis de serem detectadas em ambientes sem fio [Santana, 04]. O modo de operação especificado neste trabalho é o DCF, utilizando o CSMA/CA com reconhecimento.

Uma rede *ad hoc* pode operar no modo *hop* único (*single hop*) ou múltiplos (*multiple hops*). No modo e *hop* único, a comunicação é direta entre dois dispositivos, não havendo necessidade de determinação de uma rota. O modo de múltiplos *hops* faz-se necessário quando os dispositivos estão um pouco mais distantes, podendo haver a necessidade de determinação de uma rota [Santana, 04]. No escopo deste trabalho, todas as estações estão dentro do alcance umas das outras e o modo de operação é *hop* único.

O CSMA/CA está distribuído na camada física e na subcamada MAC (*Medium Access Control*) da camada de enlace. A camada física é responsável por escutar o meio e determinar se ele está sendo utilizado ou não através da detecção de sinal e da potência transmitida na faixa de espectro de freqüência de trabalho. Realiza, portanto, a detecção de portadora (*carrier sense* – de camada física).

No modo de operação DCF, antes de iniciar uma transmissão, uma estação sente o canal por um período de tempo randômico adicional, depois de detectar que o meio está livre. A duração deste período de duração mínima, chamado DIFS (*Distribution Coordination Function Interframe Space*), é de 34μs para o padrão 802.11a. Apenas se o meio permanecer livre por esse período de tempo adicional, a estação pode iniciar sua transmissão.

Quando a camada MAC recebe uma requisição para transmitir um quadro, uma checagem é feita através do mecanismo de detectar a portadora fisicamente e virtualmente. Se o meio não está em uso por um intervalo de DIFS, a camada MAC pode iniciar a transmissão do quadro. Se o meio está em uso durante o intervalo DIFS, a camada MAC iniciará um procedimento de *backoff* adiando assim sua transmissão por um período de tempo e também incrementará o contador de tentativa de transmissão associado a cada quadro. O Valor resultante do algoritmo de *backoff* define a quantidade de *slots* de tempo que a estação deve esperar antes de iniciar a sua transmissão. O decremento do temporizador de *backoff* é em unidades de um *slot* de tempo, caso o meio esteja livre, durante esse *slot* em particular. O decremento do temporizador de *backoff* é interrompido caso o meio fique ocupado,

reiniciando após DIFS, caso o meio voltar a ficar livre. Quando o temporizador de *backoff* alcança o valor zero, a estação pode transmitir o seu quadro. Com o envio desse quadro, a estação deve esperar um ACK para confirmar que o quadro foi recebido. Caso este ACK não retorne, assume-se que houve colisão. Então, a estação chama seus procedimentos de *backoff* e de retransmissão. Após o início da transmissão, todos os quadros devem esperar um tempo igual a 10 μs denominado de um SIFS (*Short Interframe Space*), antes de transmitir um outro quadro.

O período de *backoff* pode ser interpretado como sendo uma janela temporal na qual uma estação tem permissão para concorrer pela transmissão. Por este motivo, ele é chamado de Janela de Contenção (CW- *Contention Window*). O valor de CW varia entre um valor mínimo (*CWmin*) e um valor máximo (*CWmax*). Ele é aumentado com o número de retransmissões até um valor *CWmax*.

Outro fator importante de destaque é quanto ao tipo de estações consideradas nesse trabalho. Embora as redes sem fio e a computação móvel freqüentemente tenham uma estreita relação, elas não são idênticas [Tanenbaum, 03]. Um caso importante em que o tipo de rede considerada nesta Dissertação pode ser aplicado é aquele de empresas sediadas em edifícios, nos quais não há cabeamento de rede para conectar os computadores. Para instalar uma rede sem fio, essas empresas só precisarão adquirir alguns componentes eletrônicos, e conectar ao equipamento. Essa solução pode ser muita mais econômica do que instalar a fiação exigida pelas redes cabeadas.

Além da breve descrição do domínio do problema acima, faz-se necessário uma listagem dos requisitos funcionais (o que o sistema deve fazer) e não funcionais (características ou dimensões do sistema) para melhor entendimento das metas e objetivos a serem atingidos. A tabela abaixo possui a descrição dos requisitos funcionais e não funcionais tomando por base as especificações descritas em [Vasconcelos, 02] e em [Rocha, 02].

RF	**Descrição dos Requisitos Funcionais**
RF01	Os componentes devem permitir a construção e simulação de modelos de redes *ad hoc* sem fio padrão IEEE 802.11 tendo como referência os trabalhos de [Rocha, 02] e [Vasconcelos, 02].
RF02	Os modelos devem permitir uma ou mais fontes de tráfego.

RF03	Os modelos devem permitir uma ou mais estações emissoras.
RF04	A transmissão deve ser feita através de um canal compartilhado sem fio.
RF05	Os componentes devem oferecer propriedades que podem ser configuradas pelo especialista em redes (analista de modelagem) para viabilizar a simulação dos modelos de redes *ad hoc*.
RF06	O acesso das estações ao meio deve ser baseado no funcionamento do protocolo de acesso ao meio CSMA/CA com reconhecimento.
RF07	Durante a simulação, os componentes devem permitir coletar os seguintes dados para o cálculo de medidas de desempenho de interesse: • Número de quadros transmitidos com sucesso • Número de quadros descartados e colididos • Tempo médio de transmissão dos quadros pelo canal de comunicação • Número de quadros gerados
RNF	**Requisitos Não-Funcionais**
RNF01	Os componentes devem possibilitar, facilmente, a reutilização na construção de novas ferramentas de simulação de redes sem fio.
RNF02	Os componentes devem possibilitar, facilmente, a extensão de funcionalidades de ferramentas de simulação existentes.
RNF03	A especificação deve utilizar um processo de desenvolvimento específico para o desenvolvimento de componentes.
RNF04	A especificação deve adotar uma linguagem padrão de modelagem.
RNF05	Os componentes devem fornecer abstrações de alto nível que são facilmente utilizáveis e extensíveis.

Tabela 4.1 – Requisitos Funcionais e Não-Funcionais

4.2.1. MODELO DE PROCESSO DO NEGÓCIO

O Modelo de Processo do Negócio é um diagrama de alto nível que fornece a visão do funcionamento do negócio modelado. Este diagrama foi construído utilizando a notação de

um diagrama de atividades UML (*Unified Modeling Language*) [Larman, 98]. A Figura 4.2 mostra o fluxograma das atividades necessárias para a modelagem e simulação de uma rede sem fio *ad hoc* 802.11. Estas atividades estão associadas às fases de inicialização (Cria Modelo e Configura Modelo), à fase de simulação (Escalona Evento, Executa Simulação, Coleta Estatística) e à fase de finalização (Apresenta Resultado).

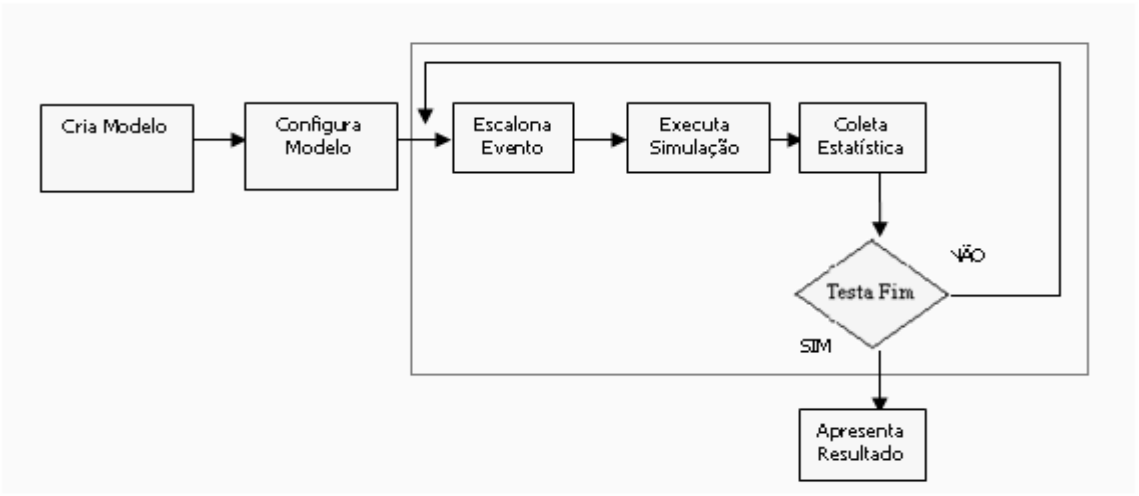

Figura 4.2 – Fluxograma das Atividades para a Modelagem e Simulação de uma Rede Local Sem Fio *Ad Hoc* 802.11

Na fase de inicialização os componentes de uma rede *ad hoc* 802.11 são criados e configurados. Nas fases de simulação e de finalização foram reutilizados componentes que fornecem o funcionamento básico de uma simulação orientada a eventos, tais como relógio simulado, geradores de variáveis aleatórias, listas de eventos, processador de medidas de desempenho, entre outros. Esses componentes foram especificados e implementados em [Rocha, 02].

Adicionalmente, apresentam-se diagramas que mostram com detalhes a dinâmica do modelo proposto nessa Dissertação. A Figura 4.3 ilustra o diagrama de atividades para o procedimento de acesso ao canal da camada MAC no padrão IEEE 802.11, utilizando o DCF básico com o protocolo CSMA/CA.

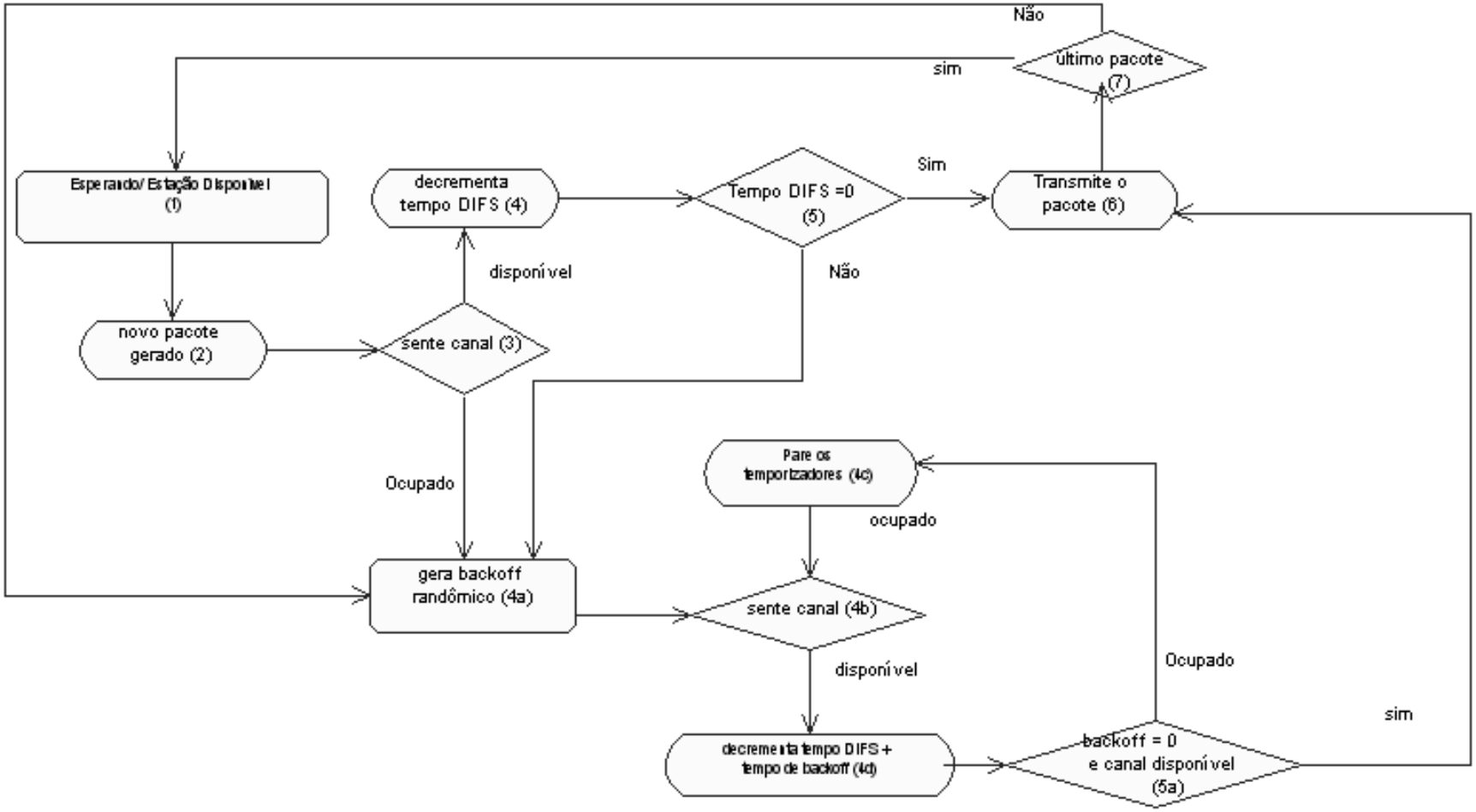

Figura 4.3 – Procedimento de acesso ao canal usando DCF básico

Como mostrado na Figura 4.3, quando uma estação tem quadros a transmitir, pode-se resumir a operação do protocolo nos seguintes passos:

1 A estação escuta o meio até que ele permaneça livre durante DIFS, logo após isso transmite o quadro. Vai para 2;

2 A estação aguarda um ACK que é enviado pelo receptor após SIFS (contando a partir do fim do envio do quadro);

2.1 Caso não receba um ACK esperado, a estação entende que houve colisão. Dobra-se o valor da janela de contenção atual (se permanecer abaixo da máxima). Vai para (3);

2.2 Caso a estação receba o ACK esperado e se ainda existirem quadros a serem transmitidos, vai para (3);

3 Um valor de *backoff* é escolhido aleatoriamente, dentro da janela de contenção atual e a estação continua escutando o meio;

3.1 A estação ao sentir o meio livre durante um intervalo de tempo igual a DIFS, inicia (ou continua) a contagem do tempo de *backoff*;

3.2 Se durante este *backoff* o meio está ocupado, a estação pára a contagem e permanece escutando o canal. Volta para (3.1);

3.3 Ao terminar o tempo de *backoff* transmite o quadro e vai para (2).

Ainda na Figura 4.3, na caixa de número 7, podemos ver que o protocolo desencoraja as transmissões requeridas por todos os quadros subseqüentes ao primeiro, tendo um *backoff* aleatório mínimo, indiferente da disponibilidade do canal. Dessa forma, o quadro de número 2 deverá esperar por um DIFS mínimo, extra ao tempo aleatório de *backoff*. A outra estação ao gerar novos quadros deverá esperar somente por um DIFS, tendo prioridade sobre a estação que possua uma série de quadros.

A Figura 4.4 ilustra os possíveis estados do meio.

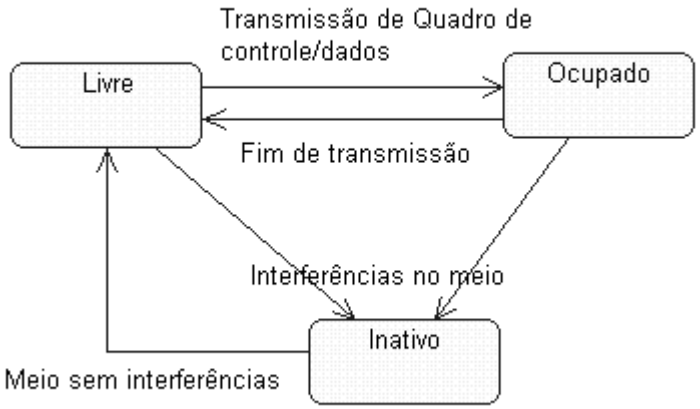

Figura 4.4 – Diagrama de estados do meio

A Figura 4.5 ilustra o diagrama de estados para o DCF. A operação começa quando um quadro está pronto para ser transmitido.

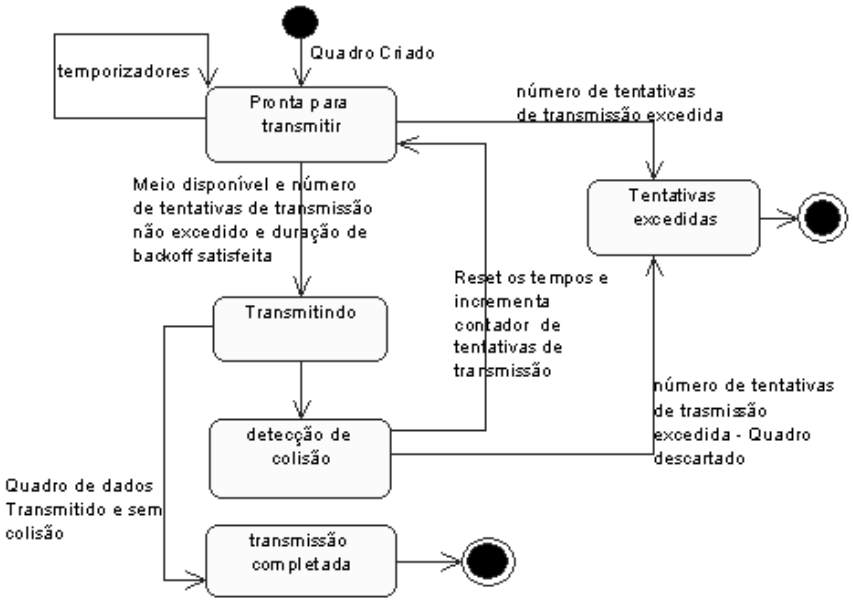

Figura 4.5 – Estados para DCF

Os estados da operação DCF são os seguintes:

- Pronto para Transmitir: este é o primeiro estado no ciclo. Este estado ocorre quando um quadro está para ser enviado. Este estado é também atingido quando uma colisão é detectada, isto é, um quadro de reconhecimento - ACK (*acknowledgement*) não é recebido pela estação transmissora.

- Transmitindo: quando no estado pronto para transmitir, se a portadora física sentir o canal ocioso e NAV = 0 (*Network Allocation Vector* – vetor de alocação de rede), lembrete interno de que a estação deve se manter inativa por um determinado período de tempo, sem o estado de tentativas excedidas e satisfizer a duração do *backoff*, este estado ocorre.
- Detecção de Colisão (sem ACK): este estado existe quando uma estação transmissora não recebe um ACK de uma estação receptora.
- Transmissão Completada: este estado ocorre quando a transmissão do quadro de dados é enviada com sucesso, sem a detecção de colisão.
- Tentativas excedidas: este estado ocorre quando o número de tentativas de transmissão do quadro ultrapassar o valor máximo estabelecido. Depois deste estado o quadro é descartado

4.2.2. DIAGRAMA DE CASOS DE USO

Os diagramas especificados nesta etapa fornecem claramente as funções que são responsabilidades do software, de pessoas ou sistemas externos ao software a partir da visão adquirida até então do funcionamento geral do software. Deve-se, portanto, identificar os atores e papéis e definir os casos de uso do sistema. Além disso, deve-se fornecer descrições narrativas (textuais) dos processos em um sistema [Larman, 98].

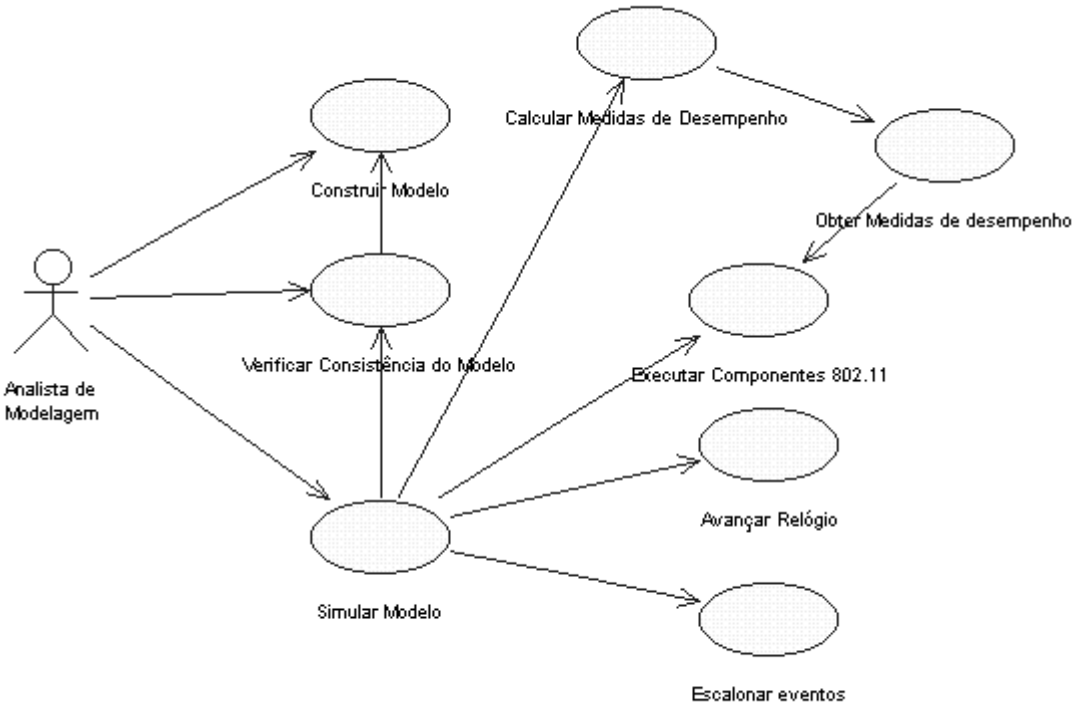

Figura 4.6 – Diagrama de Casos de Uso

A Figura 4.6 mostra o usuário interagindo com o as funcionalidades do ambiente de simulação. É importante ressaltar que as funcionalidades necessárias ao controle da execução do ambiente de simulação foram implementadas em [Rocha, 02], no qual podem ser vistos mais detalhes. Os casos de uso explorados nesta dissertação são "Construir Modelo" e "Executar Componentes 802.11".

Caso de Uso 01	**Construir Modelo**
Ator	Analista de modelagem.
Tipo	Primário
Descrição	O Analista de modelagem configura o modelo de rede *ad hoc* 802.11 que pretende simular. Os componentes 802.11 são fontes de tráfego, *hosts* (Origem e Destino), camada de acesso ao meio e o seu protocolo, sorvedouro e o meio de transmissão.

Tabela 4.2 – Descrição do Caso de Uso: Construir Modelo

Caso de Uso 02	**Executar Componentes 802.11 Para Rede *Ad Hoc***
Ator	Interação interna.
Tipo	Primário
Descrição	Os componentes são executados pelo simulador para contemplar a simulação de rede ***ad hoc***: fonte de tráfego, *hosts* (origem e destino), camada de acesso ao meio e seu protocolo, meio sem fio e sorvedouro. A fonte de tráfego gera o tráfego de informações das aplicações que rodam nos *hosts* da rede. Esse tráfego é oriundo da subcamada LLC e é repassado para a camada MAC, para estabelecer a comunicação na rede. Depois disso, a camada MAC executa o protocolo de acesso ao meio (CSMA/CA), que converte as informações em quadros e executa o acesso ao meio. Para a efetiva transmissão desses quadros, o protocolo MAC usa o meio sem fio, que representa o canal de comunicação entre os elementos na rede. Após a comunicação com sucesso entre duas estações, o *host* destino recebe o quadro, envia o ACK e repassa o quadro para o elemento sorvedouro, que representa o recebimento com sucesso do tráfego da rede.

Tabela 4.3 – Descrição do Caso de Uso: Executar Componentes *ad hoc*

4.2.3. MODELO CONCEITUAL DO NEGÓCIO

Esse modelo utiliza o diagrama de classes UML para apresentar os principais conceitos envolvidos no negócio modelado e seus relacionamentos.

Apresentamos aqui dois modelos conceituais, um que representa os conceitos que fazem parte do ambiente de simulação e outro que representa o funcionamento das redes *ad hoc* 802.11. O primeiro modelo, ilustrado na Figura 4.7, envolve os conceitos necessários para promover a simulação de rede, estende o modelo apresentado em [Rocha, 02], incluindo os componentes *ad hoc* 802.11.

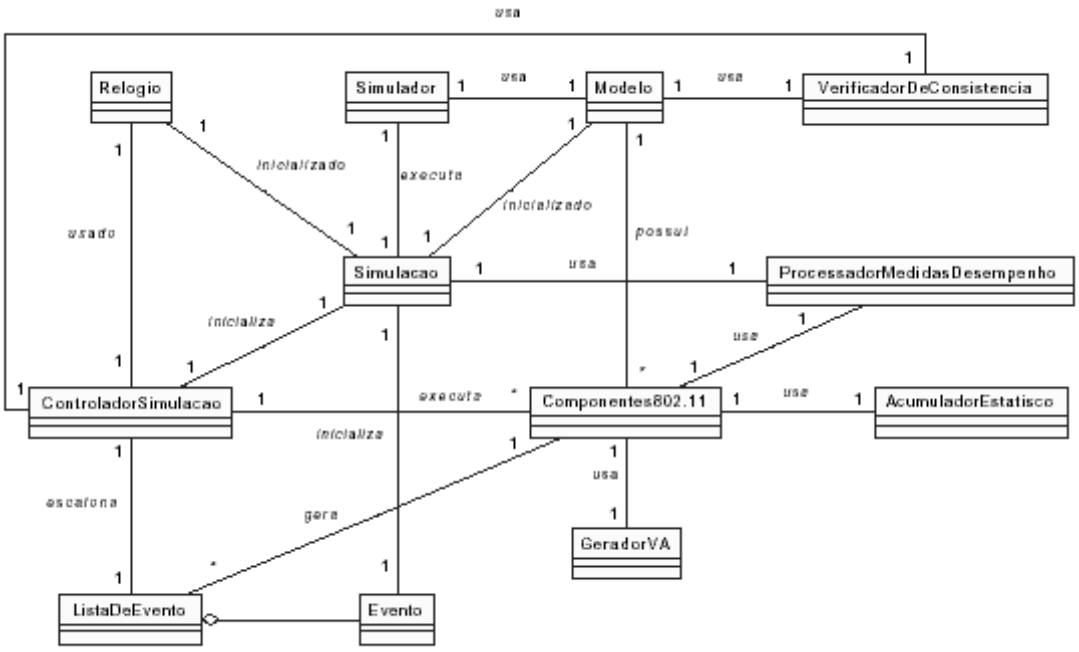

Figura 4.7 – Modelo conceitual Ambiente de Simulação

Observa-se na Figura 4.7 quais conceitos os Componentes *Ad Hoc* 802.11 se comunicam: Controlador da Simulação, Gerador de Variáveis Aleatórias (Gerador VA), Acumulador Estatístico, Processador de Medidas de Desempenho, Modelo e Lista de Eventos. A Figura 4.8 mostra os conceitos envolvidos no contexto que define os Componentes *Ad Hoc* 802.11, possíveis candidatos a componentes 802.11.

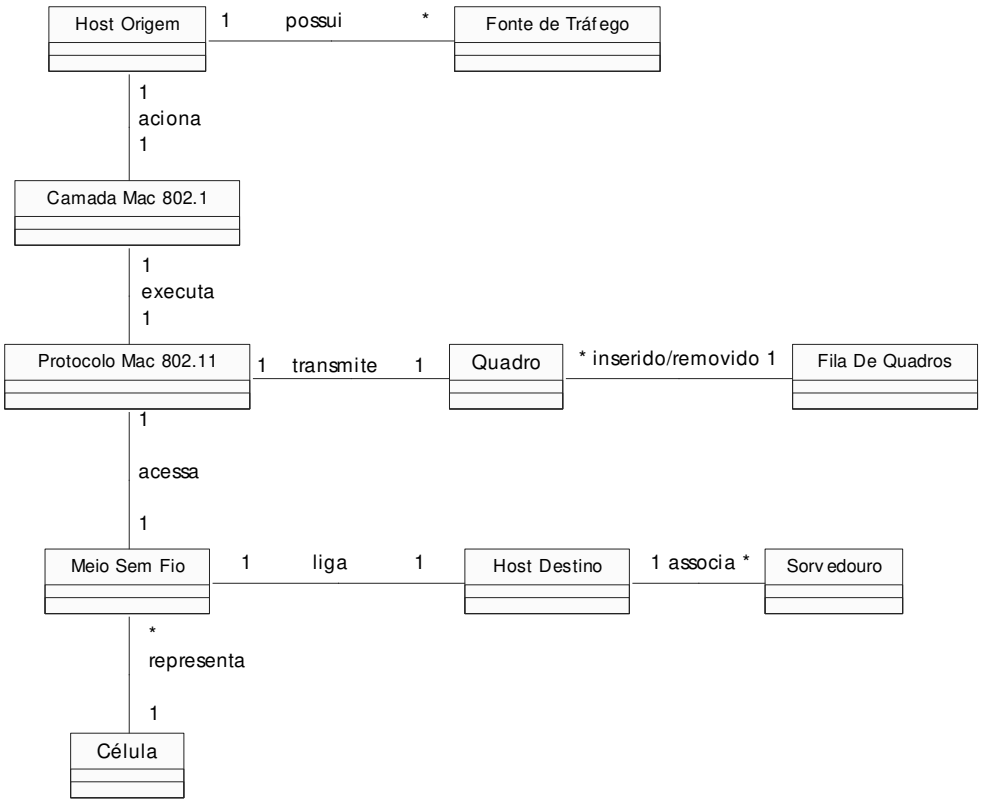

Figura 4.8 - Modelo Conceitual Candidatos a Componentes

Na Figura 4.8 podemos observar que um *host* origem pode receber tráfego de dados de várias fontes. O *host* origem aciona a sua camada de acesso ao meio, para que as informações (quadros) sejam enviadas ao *host* destino. A camada de acesso ao meio envia os quadros pelo meio de comunicação (sem fio), executando, para isso, o seu protocolo de acesso ao meio (Protocolo MAC 802.11). A transmissão com sucesso de um quadro no meio sem fio, indica que o *host* destino recebeu o referido quadro. Em seguida ao recebimento de um quadro, o *host* destino repassa esse quadro para o elemento sorvedouro, representando a entrega de quadros ao *host* destino.

4.3. Modelagem dos Componentes

Seguindo o processo de desenvolvimento Componentes UML, chega-se a fase da geração de especificações de componentes e interfaces que serão reunidas posteriormente para dar origem a uma aplicação adequada ao domínio do problema. Para alcançar este objetivo, a fase de modelagem dos componentes é subdividida em três atividades: identificação dos componentes, identificação das interações entre os componentes e a especificação dos componentes.

É importante enfatizar que o modelo de componente adotado nesta Dissertação é o *JavaBeans* [Sun, 03]. Esse modelo é composto de uma arquitetura e de uma API (*Application Programing Interface*), e possui um conjunto de regras e diretrizes que deve ser seguido pelos seus componentes, denominados *Beans.* Portanto, os diagramas que compõem essa especificação, seguem as regras definidas pelo modelo *JavaBeans* – Ver Apêndice A.

Inicialmente mostramos, através de um diagrama de seqüência, a dinâmica do ambiente de simulação no qual se inserem os componentes *Ad Hoc* 802.11 especificados. Em seguida, apresentamos diagramas de colaboração focando as interações entre os componentes 802.11 e os componentes do ambiente de simulação.

Vimos na etapa de requisitos que os modelos conceituais elaborados abordam o ambiente de simulação e os componentes *ad hoc* 802.11. Para o entendimento completo desses componentes é preciso conhecer a dinâmica de um ambiente de simulação no qual os componentes *ad hoc* 802.11 estão inseridos. Essa dinâmica pode ser ilustrada através de um diagrama de seqüência, conforme mostrado na Figura 4.9. Os componentes de um ambiente de simulação são especificados e implementados em [Rocha, 02].

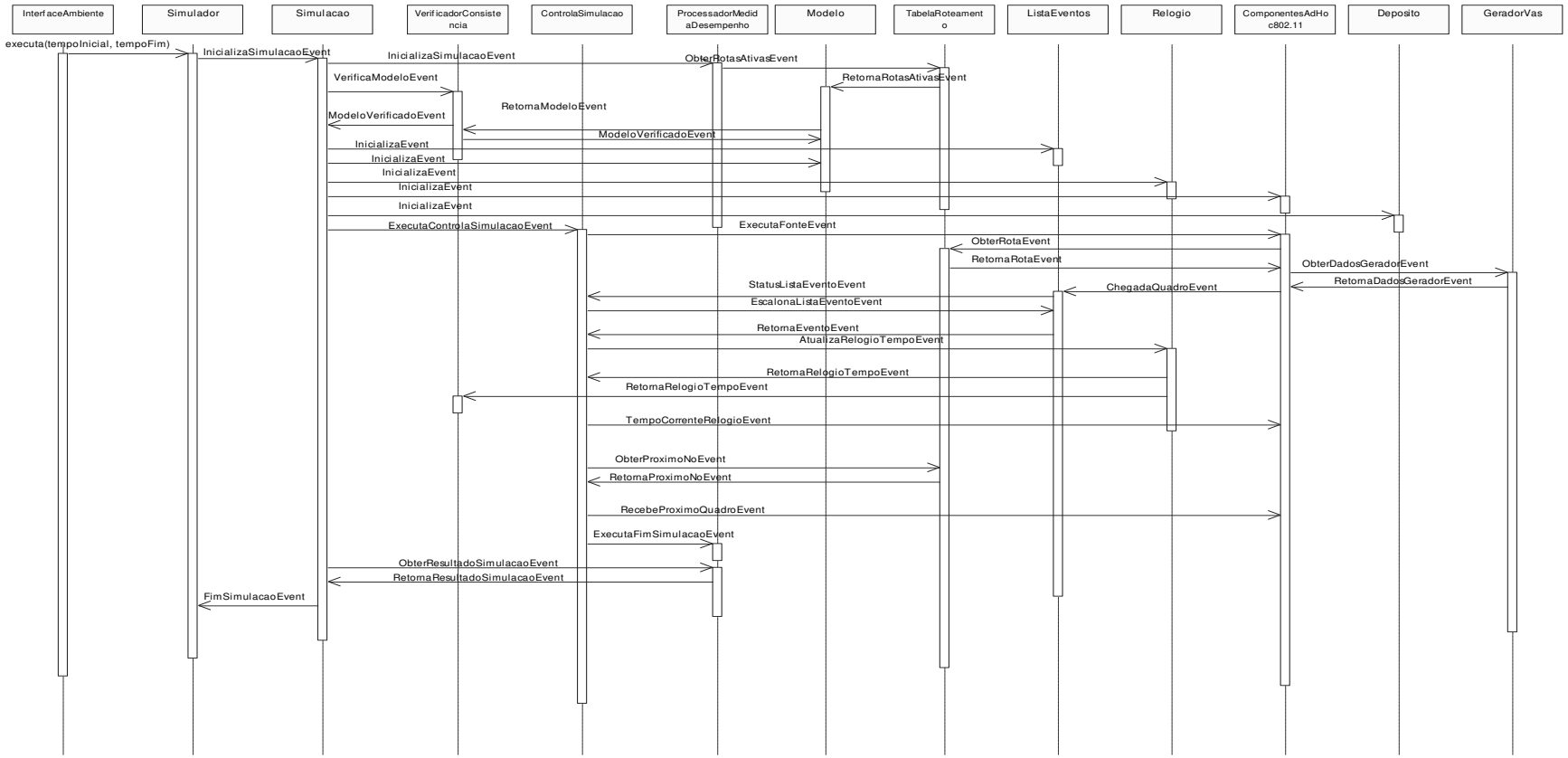

Figura 4.9 - Diagrama de seqüência Ambiente de Simulação

O diagrama de seqüência da Figura 4.9 enfatiza a etapa de execução da simulação. Portanto, é importante destacar que nesta fase quando o modelo de rede sem fio *ad hoc* 802.11 for simulado, os componentes do ambiente tais como: *Relógio, ListaEvento*s etc., já terão sido instanciados, configurados com valores iniciais e "conectados" através de eventos. Como também um modelo de rede *ad hoc* 802.11 já terá sido criado e configurado através de uma interface gráfica como mostra o diagrama de atividades da Figura 4.1.

No diagrama da Figura 4.9 podemos ver a seguinte dinâmica:

- O componente *Simulador* inicializa a simulação através do evento *inicializaSimulacaoEvent* enviado para o componente *Simulacao*.

- O componente *Simulacao* aciona o componente *VerificadorConsistencia* através do evento *verificaModeloEvent.* O componente *VerificadorConsistencia* analisa o modelo gerando o evento *modeloVerificadoEvent* para informar ao componente *Simulacao* da validação ou não do modelo.

- Em seguida, o componente Simulacao gera o evento *inicializaEvent,* que inicializa os seguintes componentes:

  - ✓ *ListaEventos* – para ativar a lista de eventos usada na simulação.
  - ✓ *ControlaSimulacao* – para escalonar os eventos e acionar componentes durante a simulação.
  - ✓ *Modelo* – para carregar no ambiente de simulação o modelo de rede que será simulado.
  - ✓ *Relogio* – para iniciar o tempo da simulação.
  - ✓ *Componentes802.11* – para carregar no ambiente os componentes de rede que são usados na simulação, ou seja, que constituem o modelo de rede que é simulado.

- Após a inicialização dos componentes supracitados, o componente *ControlaSimulacao* é acionado para iniciar o processo de escalonamento de eventos, atualização de relógio simulado e a execução das ações associadas às ocorrências dos eventos. Quando um elemento de rede é executado, dados são coletados visando à obtenção de medidas de desempenho de interesse e, eventualmente, são gerados novos eventos que são inseridos na lista de eventos.

- Por fim, o escalonamento do evento *FimSimulacaoEvent* indica que a simulação chegou ao fim. A próxima etapa consiste nos cálculos das estatísticas utilizando os dados coletados durante a simulação.

Nos diagramas de colaboração mostrados adiante fica explicitado que os componentes 802.11 interagem diretamente com diversos componentes do ambiente de simulação: *GeradorVA, ControlaSimulacao, ListaEventos* e *ProcessadorMedidasDesempenho.*

Os componentes do ambiente de simulação *ControlaSimulacao* e *ListaEventos* sempre iniciam esses diagramas. O componente *ControlaSimulacao* gera o evento *escalonaListaEventosEvent*, o qual aciona o componente *ListaEventos* que escalona o evento iminente (aquele que apresenta menor tempo de ocorrência). O evento iminente é informado ao componente *ControlaSimulacao* (geração do evento *retornaEventoEvent*). O componente *ControlaSimulacao* aciona o evento iminente (nesta Dissertação, doravante denominado evento corrente). Os diagramas de colaboração apresentados a seguir dependem do evento corrente focado e do componente que ele aciona.

A Figura 4.10 mostra a geração do evento *ExecutaControleSimulacaoEvent* pelo componente *Simulacao*. Apenas o componente *ControlaSimulacao* trata este tipo de evento. Ao "ouvi-lo", ele gera o evento *ExecutaFonteEvent* que é "ouvido" por todas as fontes do modelo. Portanto, as fontes do modelo podem gerar quadros com comprimentos fixos ou conforme amostras obtidas de uma distribuição de probabilidade definida pelo usuário. De posse da informação do *host* origem para quem deve gerar quadros, o componente *FonteTrafego* gera o evento *ChegadaQuadroEvent* que é armazenado na lista de eventos do sistema. Ao ser escalonado pelo componente *ControlaSimulacao,* o quadro associado a esse evento passa a ser associado a um outro evento chamado *RecebeQuadroEvent* que, por sua vez, aciona o componente *HostOrigem* que deve receber o quadro de acordo com o identificador. Nesse caso, cada *ChegadaQuadroEvent* ocasiona a geração de um *RecebeQuadroEvent.*

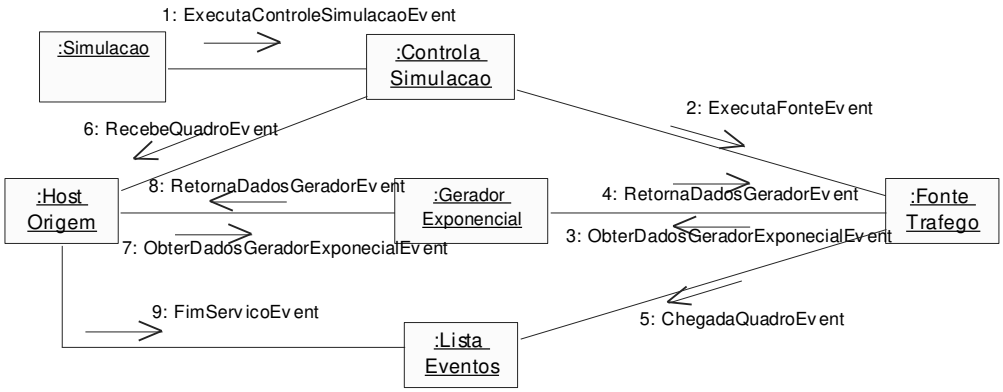

Figura 4.10 – Diagrama de Colaboração (FonteTrafego e HostOrigem)

O componente *HostOrigem* ao "ouvir" o evento *RecebeQuadroEvent* aciona a sua camada MAC e executa o seu protocolo e, quando pronta para transmitir, gera o evento *FimServicoEvent* que é "ouvido" pelo componente *ListaEventos*. O componente *ControlaSimulacao* ao escalonar o evento *FimServicoEvent*, faz acionar o componente *Enlace* através do evento *TransmiteEvent* – Figura 4.11. Neste caso, como só existe apenas um *host* apto a transmitir, o componente *Enlace* é responsável pela verificação de colisão no meio sem fio compartilhado e a correspondente contabilização dos possíveis quadros colididos. Caso não haja detecção de colisão, após a transmissão do quadro pelo enlace é gerado o evento *ChegadaQuadroEvent* que é "ouvido" pelo componente *ListaEventos*. Esse evento objetiva acionar o próximo elemento de modelagem (*HostDestino*) definido no cenário a receber este quadro. A partir da dinâmica da simulação, o *ControlaSimulacao* ao escalonar *ChegadaQuadroEvent*, gera o evento *RecebeQuadroEvent* para a respectiva *HostDestino*, essa por sua vez, gerará o evento *FimServicoEvent* que ocasionará a geração do evento *TransmiteEvent*. O componente *Sorvedouro* ao "ouvir" o evento *TransmiteEvent,* coleta as estatísticas de quadro recebido com sucesso e o descarta, gerando o evento que vai possibilitar a transmissão do reconhecimento do quadro transmitido com sucesso.

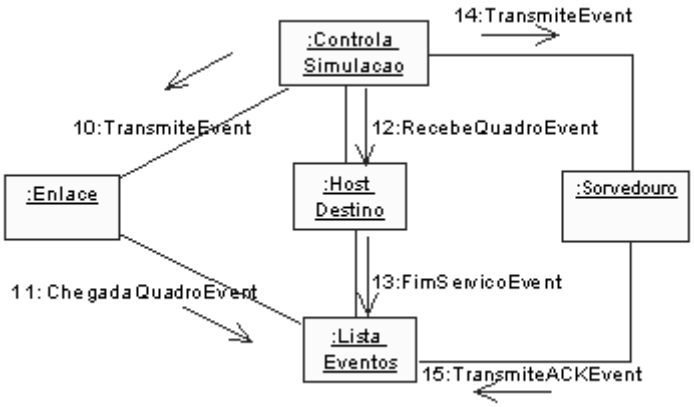

Figura 4.11 – Diagrama de Colaboração (Enlace, HostDestino e Sorvedouro)

Quando o componente *HostOrigem* não recebe o reconhecimento do quadro transmitido, caracteriza a ocorrência de colisão. Nesse caso, o componente *HostOrigem* acionará a sua camada MAC e executará o seu protocolo (incrementa o contador de tentativa de transmissão associado ao quadro, duplicação do *backoff*). Se o número de tentativa de transmissão associado ao quadro exceder o valor limite, o quadro transmitido com colisão é

retirado da fila de quadros, caso contrário, o *HostOrigem* chamará os procedimentos de retransmissão, gerando novamente o evento *RecebeQuadroEvent*, ocasionando, assim, a repetição da dinâmica da execução da simulação. Essa dinâmica se repetirá até que o evento *FimSimulacaoEvent* seja escalonado pelo *ControlaSimulacao,* que encerrará o laço que controla a execução da simulação gerando o evento *ExecutaFimSimulacaoEvent*, que acionará o componente *ProcessadorMedidasDesempenho,* para que este obtenha os valores coletados durante a simulação pelos os elementos de modelagem, como ilustra a Figura 4.12.

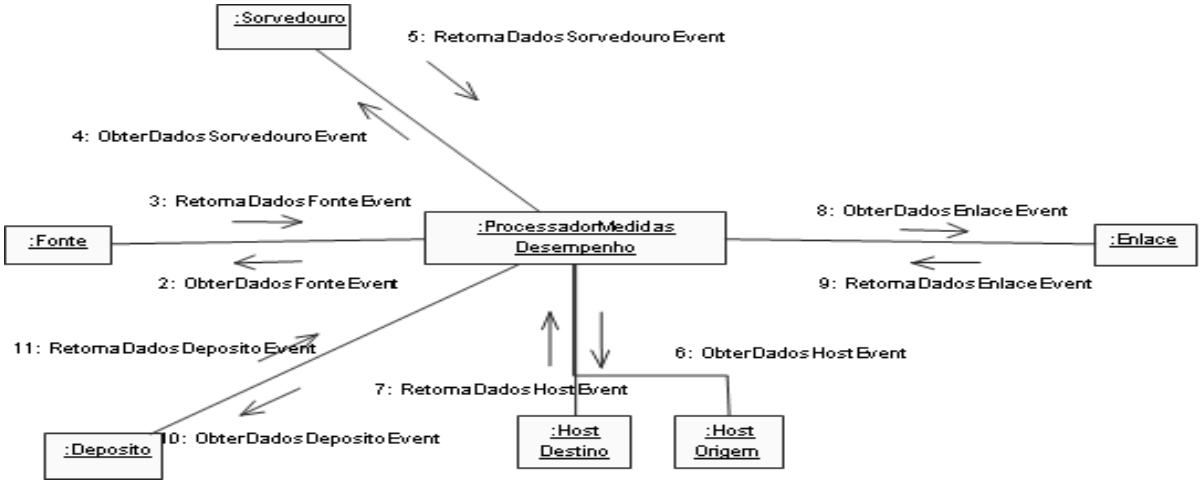

Figura 4.12 – Diagrama de Colaboração (ProcessadorMedidasDesempenho)

Cada componente é visto como uma classe, dentro dos conceitos da linguagem UML [Lula, 01]. No Apêndice B são mostrados com detalhes os diagramas de classes que representam todos os componentes *ad hoc* apresentados nesta Dissertação. Estes diagramas mostram os relacionamentos mais importantes dos componentes abordados bem como as *interfaces* e classes abstratas que implementam.

Para fins de ilustração de uma classe neste capítulo, a Figura 4.13 apresenta a classe Quadro que instancia o objeto quadro (representa o quadro de dados que transita em uma rede *ad hoc*). Mais detalhes sobre seus métodos e atributos são mostrados no Apêndice B.

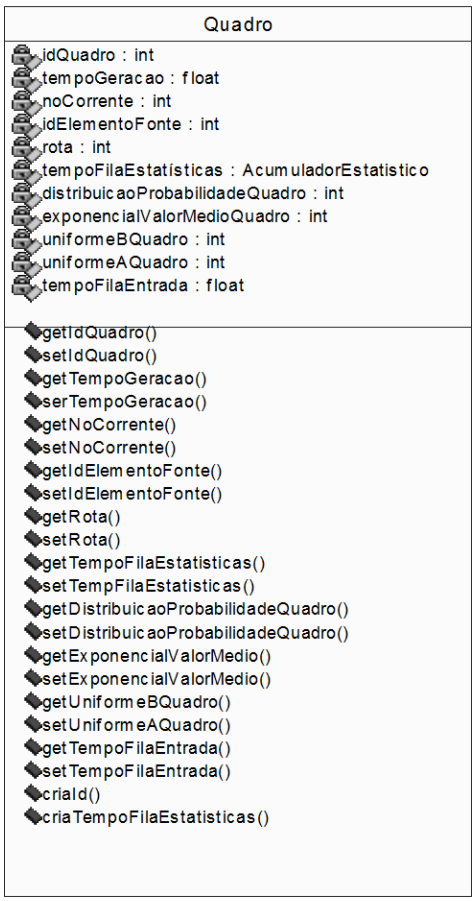

Figura 4.13 – Classe Quadro.

Capítulo 5

5. Implementação e Validação dos Componentes *Ad Hoc*

Este capítulo apresenta os principais aspectos de implementação dos componentes *ad hoc*, do ambiente de desenvolvimento para esta tecnologia, e do simulador de rede construído como estudo de caso, para fins de validação dos componentes reutilizados. Mostra-se também a validação do simulador construído, que consistiu na simulação de modelos de redes *ad hoc 802.11*, cujos resultados foram comparados com os resultados obtidos através da simulação dos mesmos modelos no ambiente de simulação *Arena*.

5.1. Materialização dos Componentes

A linguagem de programação escolhida para a implementação dos componentes foi *Java*. Esta é portável e possui uma extensa biblioteca de classes padrão, possibilitando assim um ambiente de execução independente de plataforma que permite que a aplicação seja criada uma vez e executada em qualquer lugar favorecendo a reutilização de software.

Para essa implementação foi escolhido o modelo de componentes *JavaBeans* [Sun, 03] uma vez que os componentes se comunicam através de eventos e não serão executados em um servidor. Eles são componentes de software independentes de plataforma e portáveis, uma vez que tiram proveito das vantagens que a linguagem *Java* oferece [Eckel, 00]. *JavaBeans* ou simplesmente "*Beans*" foram criados para possibilitar o desenvolvimento rápido de aplicações em *Java* montando componentes predefinidos através de uma ferramenta visual.

A comunicação dos componentes é feita através de eventos de acordo com o modelo de eventos 1.1 de *Java* [Sun, 03]. Esse modelo é baseado no conceito de classes ouvintes de eventos (*event listeners)* e fonte de eventos (*event source)*. *Listeners* são objetos interessados em receber mensagens (eventos). *Sources* são objetos que geram as mensagens (eventos). Esse modelo é descrito pelo padrão *Observer* [Gamma et al., 94].

À medida que os componentes foram implementados, testes de unidade foram realizados. Para esse fim foi usada a ferramenta de testes *Junit* [Beck & Gamma, 98].

5.1.1. CONSTRUÇÃO DE UM AMBIENTE DE SIMULAÇÃO *AD HOC* 802.11

Além da implementação dos componentes, investiu-se na utilização de um ambiente de simulação para construir simuladores de rede *ad hoc*, permitindo assim testar a integração dos componentes e validar suas implementações.

Esse ambiente de simulação de redes *ad hoc* teve como suporte o ambiente de desenvolvimento visual *Eclipse* (versão 2.0) da *IBM* [IBM, 02]. Nesse ambiente foram inseridos os componentes *ad hoc* e os componentes essenciais a um ambiente de simulação orientado a eventos, tais como: relógio simulado, geradores de variáveis aleatórias, listas de eventos, processador de medidas de desempenho, conforme propostos em [Rocha, 02]. A Figura 5.1 mostra a interface do *Eclipse,* nela pode-se observar o ambiente de simulação construído incluindo os componentes *Wi-Fi* e os componentes associados à simulação orientada a eventos.

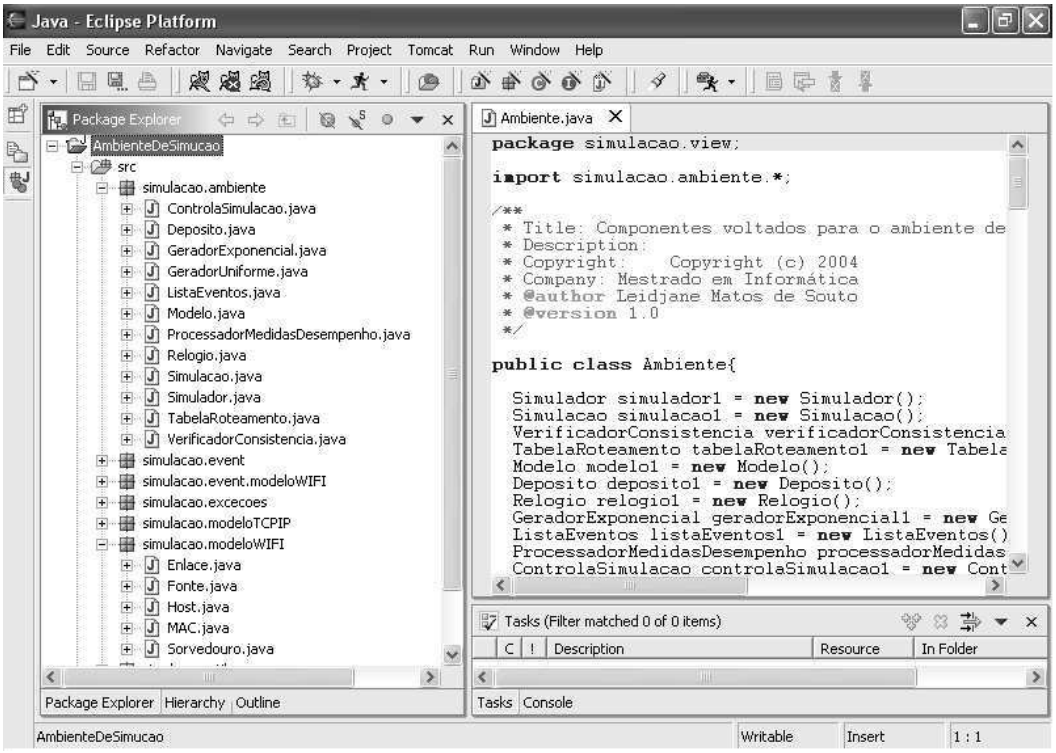

Figura 5.1 – Interface do Eclipse

Na construção desse ambiente foram implementados 4 componentes específicos para redes *ad hoc*: fontes de quadros, *hosts* origem/destino (com protocolo CSMA/CA), e

sorvedouros, totalizando 2.598 linhas de código. Adicionalmente, 16 componentes associados à simulação orientada a eventos foram reutilizados, com um total de 9.987 linhas de código. Os componentes reutilizados estão detalhados em [Rocha, 02].

5.1.2. CONSTRUINDO UM SIMULADOR PARA REDES *AD HOC* IEEE 802.11 – ESTUDO DE CASO

A implementação de um simulador para rede *ad hoc* 802.11 no ambiente de desenvolvimento mencionado pode ser feita facilmente, através da criação de uma classe *Java* chamada Ambiente.java que permite interligar os componentes. A Figura 5.2 mostra um trecho de código dessa classe que cria o simulador *ad hoc* IEEE 802.11, como estudo de caso, nesta Dissertação. Ressaltamos a facilidade desta implementação, i.e., reutilizando os componentes propostos, resume-se na classe Ambiente.java o esforço de implementar o simulador referenciado.

```
. . .
public class Ambiente{

 //Instanciação dos componentes do ambiente de simulação
 Simulador simulador1 = new Simulador();
 Simulacao simulacao1 = new Simulacao();
 VerificadorConsistencia verificadorConsistencia1 = new VerificadorConsistencia();
 TabelaRoteamento tabelaRoteamento1 = new TabelaRoteamento();
 Modelo modelo1 = new Modelo();
 Deposito deposito1 = new Deposito();
 Relogio relogio1 = new Relogio();
 GeradorExponencial geradorExponencial1 = new GeradorExponencial();
 ListaEventos listaEventos1 = new ListaEventos();
 ProcessadorMedidasDesempenho processadorMedidasDesempenho1 = new
 ProcessadorMedidasDesempenho();
 ControlaSimulacao controlaSimulacao1 = new ControlaSimulacao();
 . . .
 private void jbInit() throws Exception {

  //Cadastro dos Componentes
  //SIMULADOR
  //Cadastro para o evento CadastraEvent
  simulador1.addCadastraListener( simulacao1 );
  simulador1.addCadastraListener( tabelaRoteamento1 );
  simulador1.addCadastraListener( processadorMedidasDesempenho1 );
  simulador1.addCadastraListener( geradorExponencial1 );
  simulador1.addCadastraListener( controlaSimulacao1 );
  simulador1.addCadastraListener( listaEventos1 );
  simulador1.addCadastraListener( deposito1 );
```

Figura 5.2 - Trecho de código da classe Ambiente.java

5.1.3. MONTAGEM DA APLICAÇÃO

Com o simulador construído (exemplo mostrado na Figura 5.2), modelos de rede *ad hoc* IEEE 802.11 podem ser montados e simulados. Para facilitar a montagem e simulação de modelos, uma interface gráfica foi construída [Souto, 04].

A interface gráfica permite:

- montar os modelos (facilidades para a criação e configuração do modelo);
- definir os parâmetros necessários às simulações (início e término de cada simulação e número de replicações);
- apresentação dos resultados da simulação.

A Figura 5.3 ilustra uma janela da interface gráfica com um modelo de rede *Wi-Fi* montado. Os elementos de modelagem disponíveis nesta interface são: fontes de quadros, hosts origem/destino (implementando o protocolo CSMA/CA), o meio de transmissão sem fio e os sorvedouros (representam o recebimento de quadros no destino). Nesta versão da interface, cada elemento de modelagem recebe um número de identificação - ID.

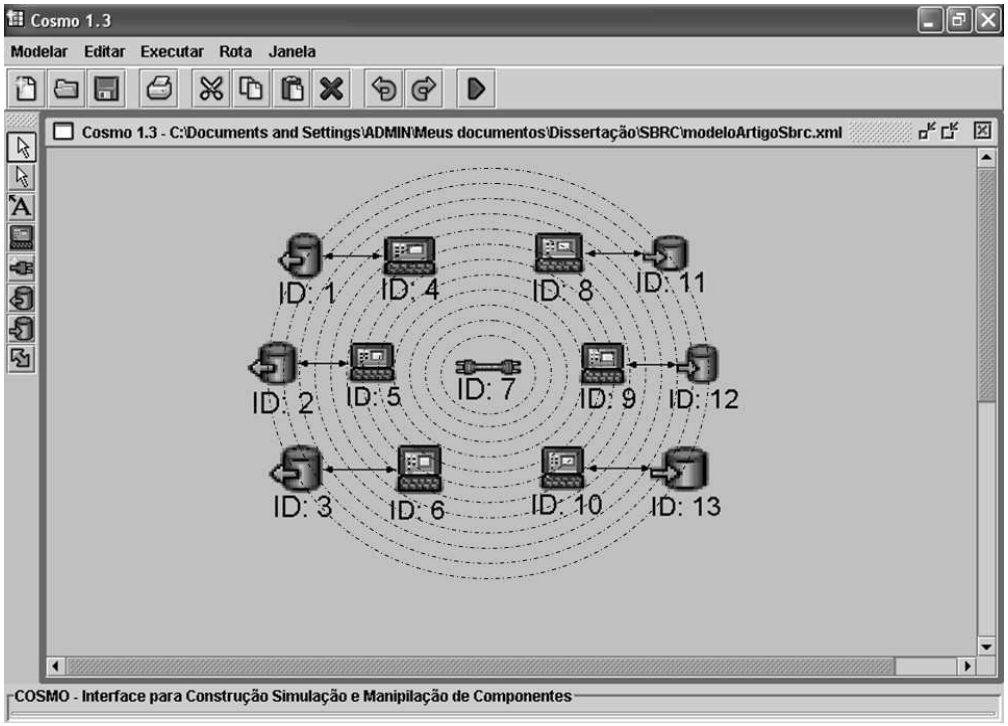

Figura 5.3 - Interface Gráfica instanciando os componentes *ad hoc* 802.11

5.2. Validação do Simulador *Ad Hoc* IEEE 802.11

Durante a implementação dos componentes *ad hoc*, a verificação de código realizada foi feita através de testes de unidade. Testes de sistema para esses componentes foram viabilizados com a construção do simulador de rede *ad hoc*, estudo de caso.

Para validar o simulador construído e, consequentemente os componentes propostos, modelos de redes *ad hoc* foram simulados e seus resultados foram comparados com os resultados obtidos através da simulação dos mesmos modelos no ambiente de simulação *Arena*.

Os modelos escolhidos para fins de validação do simulador *ad hoc* foram de dois tipos: determinístico (modelo 1) e estocástico (modelo 2). No modelo determinístico foram atribuídos valores fixos para os parâmetros de entrada dos elementos de modelagem tais como: tamanho do quadro, taxa de envio de tráfego pelas estações, capacidade do enlace, etc. No modelo estocástico foram atribuídos aos parâmetros de entrada do modelo, valores de amostras de funções de distribuição de probabilidade exponencial. Esses modelos foram também construídos e implementados no ambiente de simulação *Arena*.

Seguem os modelos simulados.

5.3. Estudos de Caso

5.3.1. MODELO 1 - DETERMINÍSTICO

A topologia do modelo a ser submetido ao simulador *ad hoc* é mostrada na Figura 5.4. O modelo foi construído através da interface gráfica já referenciada. Adotamos a convenção *uts* para unidade de tempo simulado.

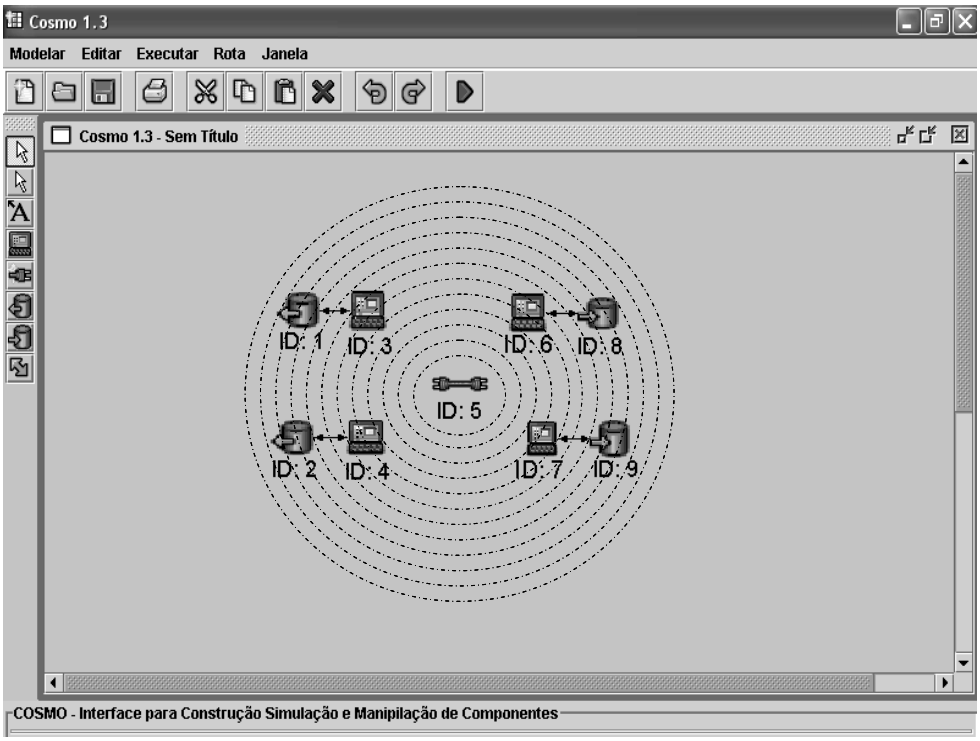

Figura 5.4 – Modelo 1 (No Simulador de Rede *ad hoc*)

O modelo da Figura 5.4 apresenta os seguintes valores de entrada, para 1 (uma) *uts* igual a 1 (um) segundo: tempo inicial da simulação igual a 0 (zero) segundos, tempo final da simulação igual 100 (cem) segundos e o número de replicação igual a 1 (um) segundo.

Em seguida, apresenta-se o mesmo modelo simulado no ambiente *Arena* usando os mesmos valores para os parâmetros de entrada. A Figura 5.5 mostra a interface do *Arena* com o modelo correspondente ao da Figura 5.4.

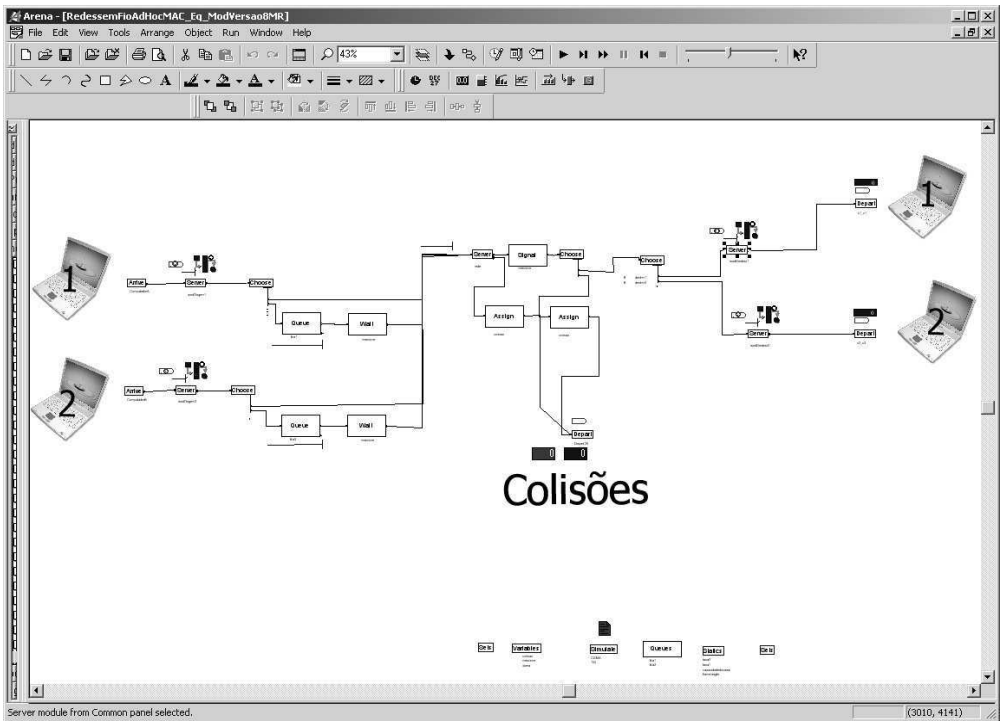

Figura 5.5 - Modelo1 (No Ambiente Arena)

Os valores dos parâmetros de entrada fornecidos pelo usuário para os elementos do modelo 1 são apresentados na Tabela 5.1.

	Tempo da Primeira Criação (*uts*)	Intervalo de Interchegada (*uts*)	Tempo de Serviço (*uts*)	Capacidade (*bits*)	Tamanho do Pacote (*bits*)	N° Max. de Pacotes
Fonte1	0	1			1000	2000
Fonte2	0	1				
HostOrigem1			2			
HostOrigem2			2			
Enlace1				2000		
HostDestino1			3			
HostDestino2			3			

Tabela 5.1 - Parâmetros de Entrada dos Elementos de Modelagem do Modelo 1 (Determinístico)

Os resultados obtidos com o ambiente *Arena* e aqueles resultados obtidos com o simulador de rede *ad hoc* são mostradas nas Figuras 5.6 e 5.7 respectivamente. Como se pode observar, as informações fornecidas pelo simulador de rede *ad hoc* são mais claras quando comparadas com aquelas fornecidas pelo ambiente *Arena.*

ARENA Simulation Results

Summary for Replication 1 of 1

Replication ended at time : 100.0

TALLY VARIABLES

Identifier	Average	Half Width	Minimum	Maximum	Observations
enlace1_R_Q Queue Time	.00000	(Insuf)	.00000	.00000	99
hostOrigem1_R_Q Queue Time	25.000	(Insuf)	.00000	50.000	51
hostDestino1_R_Q Queue Time	2.3333	(Insuf)	.00000	3.0000	33
hostOrigem2_R_Q Queue Time	25.000	(Insuf)	.00000	50.000	51
hostDestino2_R_Q Queue Time	1.4697	(Insuf)	.00000	2.5000	33

DISCRETE-CHANGE VARIABLES

Identifier	Average	Half Width	Minimum	Maximum	Final Value
enlace1_R Busy	.49000	(Insuf)	.00000	1.0000	1.0000
hostDestino1_R Available	1.0000	(Insuf)	1.0000	1.0000	1.0000
hostDestino2_R Available	1.0000	(Insuf)	1.0000	1.0000	1.0000
# in hostOrigem2_R_Q	25.000	(Insuf)	.00000	51.000	50.000
# in hostDestino2_R_Q	.48500	(Insuf)	.00000	1.0000	.00000
rede_R Available	1.0000	(Insuf)	1.0000	1.0000	1.0000
hostOrigem1_R Available	1.0000	(Insuf)	1.0000	1.0000	1.0000
hostOrigem2_R Available	1.0000	(Insuf)	1.0000	1.0000	1.0000
# in rede_R_Q	.00000	(Insuf)	.00000	.00000	.00000
# in hostOrigem1_R_Q	25.000	(Insuf)	.00000	51.000	50.000
hostDestino1_R Busy	.97000	(Insuf)	.00000	1.0000	1.0000
hostDestino2_R Busy	.97500	(Insuf)	.00000	1.0000	1.0000
# in hostDestino1_R_Q	.78000	(Insuf)	.00000	1.0000	1.0000
hostOrigem1_R Busy	1.0000	(Insuf)	.00000	1.0000	1.0000
hostOrigem2_R Busy	1.0000	(Insuf)	.00000	1.0000	1.0000
hostDestino1_R.NumberBusy	.97000	(Insuf)	.00000	1.0000	1.0000
hostDestino1_R.NumberScheduled	1.0000	(Insuf)	1.0000	1.0000	1.0000
hostDestino1_R.Utilization	. 97000	(Insuf)	.00000	1.0000	1.0000
hostDestino2_R.NumberBusy	.97500	(Insuf)	.00000	1.0000	1.0000
hostDestino2_R.NumberScheduled	1.0000	(Insuf)	1.0000	1.0000	1.0000
hostDestino2_R.Utilization	.97500	(Insuf)	.00000	1.0000	1.0000
hostOrigem1_R.NumberBusy	1.0000	(Insuf)	.00000	1.0000	1.0000
hostOrigem1_R.NumberScheduled	1.0000	(Insuf)	1.0000	1.0000	1.0000
hostOrigem1_R.Utilization	1.0000	(Insuf)	.00000	1.0000	1.0000
hostOrigem2_R.NumberBusy	1.0000	(Insuf)	.00000	1.0000	1.0000
hostOrigem2_R.NumberScheduled	1.0000	(Insuf)	1.0000	1.0000	1.0000
hostOrigem2_R.Utilization	1.0000	(Insuf)	.00000	1.0000	1.0000
rede_R.NumberBusy	.49000	(Insuf)	.00000	1.0000	1.0000
rede_R.NumberScheduled	1.0000	(Insuf)	1.0000	1.0000	1.0000
rede_R.Utilization	.49000	(Insuf)	.00000	1.0000	1.0000
hostDestino2_R_Q.NumberInQueue	.48500	(Insuf)	.00000	1.0000	.00000
rede_R_Q.NumberInQueue	.00000	(Insuf)	.00000	.00000	.00000
hostOrigem2_R_Q.NumberInQueue	25.000	(Insuf)	.00000	51.000	50.000
hostDestino1_R_Q.NumberInQueue	.78000	(Insuf)	.00000	1.0000	1.0000
hostOrigem1_R_Q.NumberInQueue	25.000	(Insuf)	.00000	51.000	50.000

COUNTERS

Identifier	Count	Limit
Sorvedouro1_C	32	Infinite
Sorvedouro2_C	32	Infinite

Simulation run time: 0.00 minutes.
Simulation run complete.

Figura 5.6 - Resultado da Simulação do Modelo 1 (Determinístico) no Arena

Resultado da Simulação
Sumário da Replicação 1 de 1

Tempo da Simulacao: 100.0

Resultados dos Acumuladores Estatísticos

Tamanho dos Pacotes no Sistema

Identificador	Mínimo	Média	Máximo	Estado	Processados
Host Origem1	1000,000	1000,000	1000,000	OCUPADO	51
Host Destino2	1000,000	1000,000	1000,000	OCUPADO	33
Host Origem3	1000,000	1000,000	1000,000	OCUPADO	51
Host Destino4	1000,000	1000,000	1000,000	OCUPADO	33

Tempos Gerais do Ambiente

Identificador	Mínimo	Média	Máximo	Observações
Tempo médio dos quadros no sistema:	5,500	36,500	67,500	32
Tempo de fila máximo no sistema	0,000	15,500	31,000	32
Tempo de fila mínimo no sistema:	0,000	0,000	0,000	32
Tempo de fila médio no sistema	0,000	10,333	20,667	32
Soma do tempo de fila sistema:	0,000	31,000	62,000	32

Tempos Coletados nos enlaces

Identificador	Mínimo	Média	Máximo	Estado	Processados
Tempo de transmissão (Enlace1):	0,500	0,500	0,500	OCUPADO	99

Tempos dos Atrasos nas Filas

Identificador	Mínimo	Média	Máximo	Pacotes em Fila	Processados
HostOrigem1					
Fila de Entrada :	0,000	25,000	50,000	50	101
Fila de Saida(Enlace1)	0,000	0,198	0,400	0	99
HostDestino1					
Fila de Entrada :	0.000	16.000	32.000	16	49

Resultados dos Contadores

Contadores Gerais

Identificador	Observações
NumeroPacotesGerados(Fonte1)	101
NumeroPacotesDestino(Sorvedouro1)	32
NumeroPacotesGerados(Fonte2)	101
NumeroPacotesDestino(Sorvedouro2)	32

Fator de Utilização

Identificador	Observações
HostOrigem1	100%
HostDestino1	97,0%
HostOrigem2	100%
HostDestino2	97,5%
Enlace1	49,0%

Simulação 1 executada com êxito

Figura 5.7 - Resultado da Simulação do Modelo 1 (Determinístico) no Simulador de Rede *ad hoc*

A Tabela 5.2 mostra um resumo dos principais resultados obtidos, comparando o fator de utilização e o número de pacotes observados. Observa-se que o desvio (diferença do valor obtido do ambiente pelo valor obtido do *Arena*) foi igual a zero para as medidas de desempenho obtidas, validando a simulação do modelo em questão.

		Ambiente de Simulação	Ambiente Arena	Desvio
Fator de Utilização	HostOrigem1	100,00 %	100,00%	0 (0%)
	HostOrigem2	100,00 %	100,00%	0 (0%)
	Enlace1	49,00%	49,00%	0 (0%)
	HostDestino1	97,00%	97,00%	0 (0%)
	HostDestino2	97,50%	97,50%	0 (0%)
N° de Pacotes Processados ou Gerados	Fonte1	101	---	---
	Fonte2	101	---	---
	HostOrigem1	51	51	0 (0%)
	HostOrigem2	51	51	0 (0%)
	Enlace1	24	24	0 (0%)
	HostDestino1	33	33	0 (0%)
	HostDestino2	33	33	0 (0%)
	Sorvedouro1	32	32	0 (0%)
	Sorvedouro2	32	32	0 (0%)

Tabela 5.2 - Resultados da Simulação do Modelo 1 (Determinístico) no Simulador de Rede *ad hoc* e no Ambiente Arena.

5.3.2. MODELO 2 - ESTOCÁSTICO

Na configuração desse modelo alguns elementos têm parâmetros de entrada com valores obtidos de uma função de distribuição de probabilidade. O ambiente oferece um componente chamado GeradorExponencial que obtém amostras de uma função de distribuição exponencial com valor médio fornecido pelo usuário. O modelo 2 é mostrado nas Figuras 5.8. e 5.9, respectivamente, no simulador de rede *ad hoc* e no ambiente *Arena*.

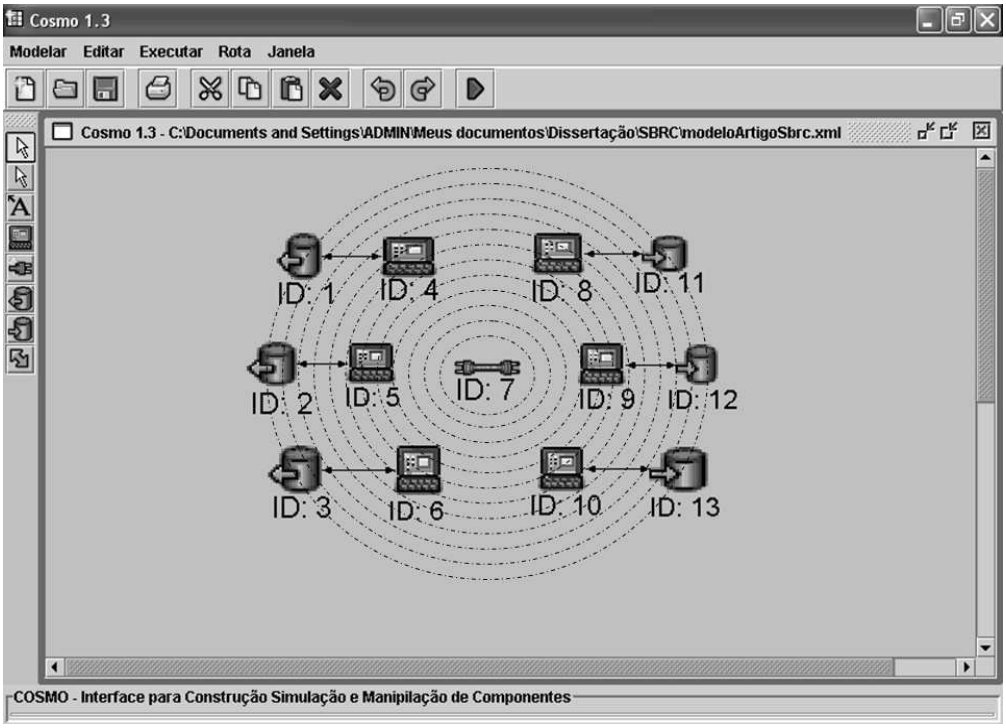

Figura 5.8 – Modelo 2 (No Simulador de Rede *ad hoc*)

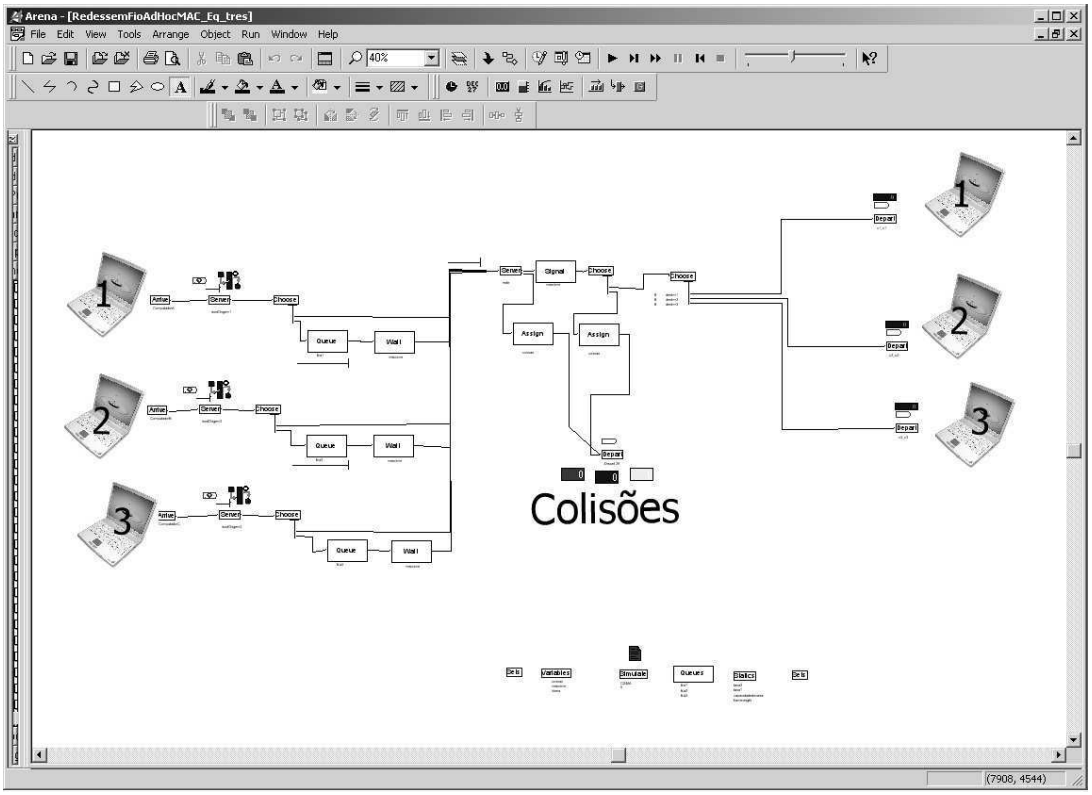

Figura 5.9 – Modelo 2 (No Ambiente *Arena*)

Nesse modelo, tem-se uma rede *ad hoc* 802.11 com seis *hosts* em área de mesmo alcance. Considera-se 1 (um) *uts* igual a 1 (um) segundo. São as seguintes as especificações para o modelo: (i) capacidade de transmissão do meio é de 2 Mbps; (ii) quadros são gerados pelas fontes 1, 2 e 3, tráfego variável, com taxas conforme distribuição exponencial com média 4 quadros por segundo; (iii) assume-se que os quadros têm comprimento 64 bytes; e os *hosts* 4, 5 e 6 transmitem para os *hosts* 8, 9 e 10, respectivamente, formando assim três pares de estações com comunicação *peer-to-peer*. Quanto aos parâmetros de simulação, considera-se que a simulação começa no tempo "0" (zero) e finaliza no tempo 5 (cinco) segundos, com 10 (dez) replicações independentes. Para fins de validação, mostramos na Tabela 5.3 as medidas de desempenho obtidas: número médio de quadros recebidos pelos elementos *Hosts* Destinos e número médio de quadros colididos por *Host* Origem, para 10 (dez) simulações realizadas, no ambiente *ad hoc* e no ambiente *Arena*. Comparando os resultados obtidos entre o simulador de rede *ad hoc* e o ambiente *Arena* verifica-se que os desvios (desvio = |A-B|/A, onde A refere-se ao resultado de uma medida de desempenho obtida através do simulador *ad hoc* e B refere-se ao resultado desta medida obtida através do simulador no ambiente *Arena*) ficaram abaixo do valor 3 (três)% sendo, portanto, aceitáveis.

	Média Quadros Recebidos	IC**	Desvio (|A-B| /A)
Host Destino1 (ID*=8)	993,9	(± 19,6)	1,59%
Host Destino2 (ID =9)	992,9	(± 28,4)	2,40%
Host Destino3 (ID=10)	997,1	(± 14,5)	1,11%
	Média Quadros Colididos por Origem	IC	Desvio (|A-B| /A)
Host Origem1 (ID=4)	19,8	(± 4,7)	1,50%
Host Origem2 (ID =6)	20,5	(± 3,8)	2,87%
Host Origem3 (ID=6)	20	(± 4,2)	1,00%

*ID: Identificador do elemento no modelo ** IC: Intervalo de Confiança com nível de confiança igual a 95%

Tabela 5.3 - Número Médio de Quadros Recebidos e Colididos pelos *Hosts* Destino

5.3.3. CONSIDERAÇÕES FINAIS

No modelo determinístico os resultados obtidos através do simulador de rede *ad hoc,* para as medidas de desempenho (número de quadros transmitidos com sucesso; número de quadros descartados e colididos; e tempo médio de transmissão dos quadros pelo canal de

comunicação e número de quadros gerados), foram iguais àqueles obtidos pelas simulações no ambiente *Arena*. Para o modelo estocástico, foram observados que os desvios referentes aos resultados observados eram aceitáveis (menores que 3%), validando o simulador implementado.

Capítulo 6

6. Conclusões e Trabalhos Futuros

Este trabalho apresentou componentes de software voltados para a construção de simuladores de redes *ad hoc* 802.11, utilizando a metodologia orientada a componentes *(Componentes UML*).

A adoção de uma metodologia orientada a componentes, favorece a reutilização de software, possibilitando acrescentar novas funcionalidades, como também facilita a criação de novos componentes. As especificações geradas tiveram como ponto de partida as especificações apresentados em [Rocha, 002] e [Vasconcelos, 02]. A tecnologia para a implementação dos componentes escolhida foi *Java* (*JavaBeans*) [Sun, 03] e o Ambiente de Desenvolvimento utilizado foi *Eclipse* da [IBM, 02].

Os usuários que podem reutilizar os componentes são os desenvolvedores de simuladores que podem construir simuladores próprios, simplesmente configurando e "conectando" os componentes propostos, estendendo suas funcionalidades, ou mesmo, adicionando novos componentes. Os componentes aqui apresentados, embora sejam voltados para as redes *ad hoc* podem também ser reutilizados e estendidos para outras tecnologias de redes sem fio. Com um simulador construído, usuários especialistas em rede, podem rapidamente montar e submeter graficamente seus modelos para fins de estudo de desempenho dos mesmos.

Nesta Dissertação, investiu-se na construção de uma ambiente de simulação para a construção de ferramentas de simulação de redes *ad hoc*. Este sistema permitiu, com facilidade, construir um simulador de rede *ad hoc* IEEE 802.11, como estudo de caso, validando a reutilização dos componentes *ad hoc*. Uma interface gráfica também foi construída para facilitar a submissão de modelos neste simulador. A validação do simulador consistiu na comparação dos resultados de simulações de modelos de redes *ad hoc* obtidos através do ambiente de simulação desenvolvido nesta Dissertação com os resultados obtidos no ambiente de simulação *Arena*.

Durante a implementação, a verificação de código realizada foi feita através de testes de unidade. A construção do simulador *ad hoc* permitiu a realização de teste de sistema, comprovando a funcionalidade dos componentes. Os resultados das simulações dos modelos apresentados no capítulo 6 mostraram que os requisitos funcionais, apresentados na fase de especificação, foram atendidos e que os componentes *ad hoc* foram implementados corretamente. Esses resultados também validaram o simulador construído.

Trabalhos futuros podem investir na implementação de componentes adicionais que permitam a simulação de modeles de redes infra-estruturadas 802.11, podendo dessa forma se implementar outros tipos de protocolos de acesso ao meio, como o PCF e a extensão do CSMA/CA para o tratamento de estações ocultas (*hidden node – HN*), bem como se acrescentar novas funcionalidades aos componentes propostos como funções de mobilidade, tratamento de estações perdidas e etc.

Protocolos de roteamento podem ser implementados permitindo que modelos de rede *ad hoc multi-hop* possam ser construídos e simulados.

Pode-se investir também no refinamento da interface gráfica com intuito de facilitar a análise do desempenho dos elementos de modelagem durante e após a simulação, através de gráficos e outros elementos visuais.

Referências Bibliográficas

[Almeida, 99] Almeida, Marcelo, J. S. C.; **"***ATMLib – Uma Biblioteca de Classes para a Construção de Simuladores de Redes ATM: Proposta e Implementação*", Dissertação de Mestrado, CCT, Universidade Federal da Paraíba, Campina Grande, novembro de 1999.

[AltaGroup, 96] Alta Group, *"BONES Designer User's Guide"***,** Alta Group of Cadence Design System, Inc., 1996.

[Atkinson, 01] Atkinson, Colin, Joachim Bayer, Christian Bunse, Erik amsties, Oliver Laitenberger,Roland Laqua, Dirk Muthig, Barbara aech, Juergen Wuest, and Joerg Zettel. "*Component-based Product ine Engineering with UML"*. Component Software Series. Addison-Wesley, 2001.

[Banks, 96] Banks, J., Carson, J., Nelson, B; *"Discrete-Event System Simulation"*. Prantice Hall, Inc 1996, p.548.

[Beck & Gamma, 98] BECK, Kent, GAMMA, Erich. "*Test Infected: Programmers Love Writing Testes"*, 1998, Disponível em: http://www.junit.org/.

[Cheesman, 01] Cheesman, John and John Daniels. "*UML Components: A Simple Process for Specifying Component-Based Software*". Addison-Wesley, 2001.

[Crow, 97] Crow B.P., I. Widjaja, J.G.Kim e P.T. Sakai. "*IEEE 802.11 wireless local area networks*", IEEE Communications Magazine, vol.35, no.9, pp. 116-126, setembro 97.

[D'Souza, 98] D'Souza Desmond, F.; Wills, Alan C.; *"Objects, Components and Frameworks with UML: The Catalysis Approach"*; Addison-Wesley, 1998.

[Damoun, 79] Damoun, F., Kleinrock, L.; *"Stochastic Performance Evaluation of Hierarchical Routing for Large Networks",* Computer Networks, vol. 3. Pp. 337-353, novembro de 1979.

[Dias, 92] Dias, Maria Madalena; **"***SIMILE - Um Simulador Reutilizável para Avaliação de Desempenho de Redes Locais"* Dissertação de Mestrado, CCT, Universidade Federal da Paraíba, Campina Grande, abril de 1992.

[Eckel, 00] Eckel, Bruce; "*Thinking in Java*", 2a edição, Revision 12, New Jersey, Prentice-Hall, 1978.

[Englander, 97] Englander, Robert. *"Developing Java Beans"*, O'Reilly & Associates, Inc., 1997.

[Fowller, 97] M. Fowller and K. Scott; "*UML Distilled – Applying the Standard Object Modeling Language*", Addisson-Wesley, January, 1997.

[Freire, 00] Freire, Raissa Dantas; "*Especificação de um Framework baseado em Componentes de Software Reutilizáveis para Aplicações de Gerência de Falhas em Redes de Computadores*"; Dissertação de Mestrado, CCT, Universidade Federal da Paraíba, Campina Grande, abril de 2000.

[Gamma et al., 94] Gamma, E.,Helm R., Johnson R., Vlissides J.; "*Design Patterns: Elements of Reusable Object-Oriented Software*", Addison-Wesley, Massachussets, USA, 1995.

[Garcia, 03] Garcia, Francilene; *"Desenvolvimento Global de Software"*, Disciplina do curso de Pós Graduação em Informática; Universidade Federal de Campina Grande; Material on-line: www.dsc.ufcg.edu.br/~garcia/cursos/dglobal_software; período 2º semestre, ano 2003.

[GloMoSim, 98] X.; Bagrodia, R.; Gerla, M. *"GloMoSim: a Library for Parallel Simulation of Large-scale Wireless Networks"*, Proc. of the 12th Workshop on Parallel and Distributed Simulations -- PADS '98, May 26-29, 1998.

[Howell, 97] Howell, Fred; McNab, Ross; "*Simjava: A Discrete Event Simulation Package for Java with Applications in Computer Systems Modelling*"; Department of Computer Science –The University of Edinburgh, Scotland – UK, 1997.

[IBM, 02] IBM. "*Eclipse Project*", 2002, disponível em: http://www.eclipse.org/platform.

[IEE802.11, d] IEEE Doc. IEEE P802.11-96/49C. "*802.11 Tutorial – 802.11 MAC Entity: MAC Basic Access Mechanism Privacy and Access Control*".U.S.A., 1999.

[IEEE802.11, a] "*Wireless LAN medium access control (MAC) and physical layer (PHY) specifications*". IEEE Standard 802.11, 1999.

[IEEE802.11, b] "*Wireless LAN medium access control (MAC) and physical layer (PHY) specifications: Higher-speed physical layer extension in the 2,4Ghz band*" IEEE Standard 802.11b, 1999.

[IEEE802.11, c] "*Wireless LAN medium access control (MAC) and physical layer (PHY) specifications: Higher-speed physical layer in the 5Ghz band*" IEEE Standard 802.11a, 1999.

[INFO, 03] INFO EXAME: **"***Banda Larga*". Ano 18. Nº212. Novembro/2003. 60p.

[INFO, 04] INFO EXAME: "*Redes Wi-Fi*," Nº218, Maio/2004, 42p.

[Kelton, 98] Kelton, W. David**; "***Simulation with arena"*, WCB/McGraw-Hill; 1998.

[Kelvin, 01] Kelvin, Lopes Dias., Sadok, Djamel Fauzi Hadj. "*Internet Móvel: Tecnologias, Aplicações e QoS"*. Dissertação de Mestrado, UFPE, 2001.

[Kronbauer, 98] Kronbauer, Artur H.; **"***Avaliação de Protocolos Multicast em Redes TCP/IP",* Dissertação de Mestrado, CCT, Universidade Federal da Paraíba, Campina Grande, junho de 1998.

[Larman, 98] Larman, C;Applying *"UML and Patterns – IN Introduction to Object-Oriented Analysis and Design"*; Prentice-Hall, 1998.

[Lee, 01] LEE, Edward A. *"Overview of the Ptolemy Project"*, Technical Memorandum UCB/ERL M01/11, University of California, Berkeley, EUA. Março de 2001. Disponível em http://ptolemy.eecs.berkeley.edu.

[Loureiro, 03] Loureiro, Antônio A. F. e et al. "*Comunicação Sem Fio e Computação Móvel: Tecnologias, Desafios e Oportunidades*". Congresso SBC 2003.

[Lula, 01] Lula, Juliana C. L de A.; "*Especificação de Componentes para um Ambiente de Simulação de Redes TCP/IP"***,** Projeto de Dissertação de Mestrado, Pós-Graduação em Informática, UFPB, 2001.

[MobiCS, 02] MobiCS, "*Mobile Computing Simulator*". Disponível em: http://www.inf.puc-rio.br/~endler/MobiCS/. Acesso em jun/2002.

[MobiCS2, 03] MobiCS2, "*Um Framework para a Simulação de Redes Móveis Ad hoc*". Disponível em: http://www.inf.puc-rio.br/~endler/MobiCS/. Acesso em agos/2003.

[ModSim-II, 89] MODSIM-II; "*MODSIM-II: An Object-Oriented Simulation Language for Sequencial and Parallel Processors*", Proceedings of the 1998 Winter Simulation Conference, Piscataway, NJ, pp 172-189, 1989.

[NS, 03] NS, "*Network Simulator*", Projeto VINT, E.U.A. Disponível em: http://www.isi.edu/nsnam/ns/ . Acesso em agos/2003.

[Opnet, 02] Opnet. "*Opnet Modeler*". Disponível em: http://www.opnet.com/products/modeler/home.html

[Parsec, 98] R. et al Parsec: "*A Parallel Simulation Environment for Complex Systems"*, Computer, Vol. 31(10), October 1998, pp. 77-85.

[Pegden, 95] Pegden, C., Shannon, R., Sadowski, R.; "*Introduction to Simulation Using SIMAN*". McGraw-Hill, Inc, 1995.

[Roberts, 94] Roberts, Chell; Dessouky, Yasser; "Na Overview of Object-Oriented Simulation"; SIMULATION, Number 70, pp. 359-368, june, 1998.

[Rocha, 02] Rocha, Flávio Gonçalves. *"Implementação e Validação de Componentes para a Construção de Ambientes de Simulação de Redes TCP/IP"*, Dissertação de Mestrado, UFPB, 2002.

[Rubinstein, 02] Rubinstein, Marcelo G; *"Qualidade de Serviço em Redes 802.11"*. SBRC 2002.

[Santana, 04] Santana, André A., Schweitzer, Christiane M., Cavalho, Tereza C. M. de B., Venturini, Yeda R. "*Novos Paradigmas de Projeto e Soluções de Segurança e Desempenho em Redes Sem Fio*", SBRC 2004.

[Sauvé, 03] Sauvé, Jacques; *"Análise e Projeto de Sistemas Orientados a Objeto"*, Disciplina do curso de Pós Graduação em Informática; Universidade Federal de Campina Grande; Material on-line: www.dsc.ufcg.edu.br/~jacques/cursos/2003.1/poo; período 1º semestre, ano 2003.

[Silva, 00] Silva, Ricardo Pereira e.; *"Suporte ao Desenvolvimento e Uso de Frameworks e Componentes*"; Tese de Doutorado, UFRGS/II/PPGC, Porto Alegre: março de 2000.

[Silva, 01] Silva, Karina Rocha Gomes da; "*Avaliação de desempenho do Buffer Multiclasse COMATM"*. Dissertação de Mestrado, UFPB, 2001.

[Silva, 99] Silva, Ricardo Pereira e., Price, Roberto Tom *"Suporte ao desenvolvimento e uso de componentes flexíveis";* Anais do XIII Simpósio Brasileiro de Engenharia de Software. Florianópolis, SC. Outubro de 1999.

[Soares, 92] Soares, Luiz F. Gomes ; *"Modelagem e Simulação Discreta de Sistemas";* VII Escola de Computação, São Paulo, 1992.

[Soares, 95] Soares, Luiz Fernando G. "*Redes de Computadores: das LANs, MANs às redes ATM / Luiz Fernando Gomes Soares, Guido*

Lemes, Sérgio Colcher". Rio de Janeiro, Campus, 1995.

[Souto, 04] Souto, Leidjane e et al. "*Cosmo – Uma Interface Gráfica para Construção de Simuladores de Redes*". RT DSC/CCT/UFCG/001/05.

[Sun, 03] Sun Microsystems Inc; "*Java Beans Specification*". Disponível em http://www.sun.com/beans. Acesso em nov/ 2003.

[Szyperski, 99] Szyperski, Clemens; "*Component Software Beyond Object-Oriented Programming"*, Addison-Wesley, 1999.

[Takus, 97] Takus, David A. *"Arena Software Tutorial"***,** Pensylvania, System Modeling Corporation, 1997.

[Tanenbaum, 03] Tanenbaum, A; "*Redes de Computadores*"; Editora Campus; 4ª edição, 2003.

[Torres, 01] Torres, Gabriel. "*Redes de Computadores: Curso Completo*". Axcel Books 2001.

[Vasconcelos, 02] Vasconcelos, Geovane Vitor. "*Especificação de Componentes para Modelagem de Redes Locais de Computadores Sem Fio Padrão IEEE 802.11"*, Dissertação de Mestrado, UFPB, 2002.

[Wagner, 00] Wagner, Marcus Vinícius da Silva. *"Especificação de Componentes para a Simulação de Redes TCP/IP"*, Dissertação de Mestrado, UFPB, 2000.

[Zanetti, 99] Zanetti, Alberto René. "*Redes Locais Sem Fio / Alberto René Zenetti, Leandro de Carvalho Gonçalves*". Pesquisa de Mestrado, UFSCar, 1999.

Apêndice A

7. JavaBeans

Um *bean* é um componente de software reutilizável, baseado na especificação *JavaBeans* da Sun, que pode ser manipulado visualmente em uma ferramenta construtora [Sun, 03].

JavaBeans é uma arquitetura que é utilizada para construir componentes em Java [Englander, 97]. Essa arquitetura suporta características de reusabilidade de software e orientação a objetos. Uma das mais importantes características de *JavaBeans* é que ela não modifica a linguagem Java. Em Java, *bean* significa componente. A API utilizada para a construção de componentes de software em Java chama-se `java.beans`.

Os principais aspectos de um modelo de um *bean* são [Szyperski, 99]:

- *Eventos*: os *beans* podem ser fontes ou consumidores potenciais de eventos. É através da troca de eventos que os *beans* se comunicam uns com os outros.A ferramenta de composição conecta consumidores a fontes.
- *Métodos*: serviços que o cliente pode utilizar.
- *Propriedades*: As propriedades podem ser modificadas em tempo de composição e representam os atributos de estado e de comportamento através dos quais *beans* podem ser configurados. As mudanças nas propriedades podem disparar eventos.
- *Introspecção*: um *bean* pode ser inspecionado pela ferramenta de composição para encontrar suas propriedades, eventos e métodos suportados.
- *Configuração*: usando a ferramenta de composição, um *bean* pode ser configurado modificando suas propriedades.
- *Persistência*: os *beans* configurados e conectados necessitam ser salvos para futuro carregamento em tempo de execução da aplicação.

7.1. Eventos

Os eventos são objetos criados por uma fonte de eventos (*event source*) e propagado para todos os consumidores de eventos (*event listeners*) cadastrados. Os consumidores se registram em fontes de eventos para receber notificações de eventos. Para se registrar em uma fonte, os consumidores usam métodos padronizados, cujas assinaturas são:

```
public void add<tipo>Listener (<tipo> Listener)

public void remove<tipo>Listener (<tipo> Listener)
```

O pacote `java.util` provê o suporte básico para o modelo de eventos dos *beans*. A comunicação baseada em eventos geralmente é *multicast*.

7.2. Propriedades

O *bean* pode definir um número arbitrário de propriedades. Uma propriedade é um atributo ou característica do *bean* que pode afetar sua aparência ou comportamento. Propriedades são referenciadas por seu nome e pode ter qualquer tipo, incluindo tipos primitivos, tal como *int*, e tipos de classes ou interfaces, tal como `java.awt.Color`. Propriedades são geralmente parte do estado persistente de um objeto.

As propriedades de um *bean* são acessadas através de métodos padronizados para ler e escrever. As propriedades podem ser para ler/escrever, somente para ler ou somente para escrever. Os métodos usados para acessar as propriedades seguem o padrão de projeto para propriedades [Englander, 97]. Suas assinaturas são:

```
public void get<NomePropriedade> (<tipo> valor);

public void set<NomePropriedade> (<tipo> valor);
```

7.3. Intropescção

Quando se usa uma ferramenta de desenvolvimento visual (exemplo *BeanBox*), ela deve expor as propriedades, métodos e eventos do *bean*, permitindo manipular sua aparência e comportamento. Para descobrir essas priopriedades e eventos, a ferramenta usa um mecanismo de introspecção, realizado através da API de reflexão Java, `java.lang.reflect`, que permite descobrir características de um *bean* via padrões de projeto.

Para que este mecanismo de introspecção funcione, programadores de *beans* devem seguir convenções de nomeação [Englander, 97]. Essa padronização é uma combinação de

regras para a formação de assinatura de métodos, seu tipo de retorno e seu nome. Por exemplo, para as propriedades, deve existir os métodos de acesso *get* e *set*, descritos nesta seção.

Algumas vezes o mecanismo de reflexão força a utilização de padrões de projeto que não são necessários ou, então, deseja-se expor informações de *beans* que não podem ser representados pelos padrões de projeto. Nestes casos, o *bean* deve fornecer, explicitamente, informações sobre suas propriedades, métodos e eventos, implementando a interface `java.beans.BeanInfo`. Essa interface especifica um conjunto de métodos que podem ser usados para recuperar vários elementos de informação sobre o *bean*. Se essa interface for implementada, a ferramenta de composição vai usá-la para descobrir os métodos, propriedades e eventos do *bean*.

7.4. Persistência

A maioria dos componentes mantém informação que definem sua aparência e comportamento. Essa informação é conhecida como o estado do objeto. Algumas dessas informações são representadas pelas propriedades do objeto. Quando uma aplicação é executada, os componentes devem automaticamente exibir um comportamento pré-estabelecido. Portanto, a informação de estado de todos os componentes precisa ser salva em algum meio de armazenamento persistente, visto que ele pode ser usado para recriar todo o estado da aplicação em tempo de execução.

A arquitetura JavaBeans usa o mecanismo de serialização do Java para a persistência. Tudo que o *bean* deve fazer é implementar a interface `java.io.Serializable`.

Componentes Java compilados são empacotados em arquivos do tipo JAR. Os arquivos JAR podem conter um número arbitrário de arquivos e podem prover compressão baseada no formato ZIP. Esses arquivos podem ser usados para empacotar arquivos de classes relacionados, *beans* serializados, e outros recursos necessários ao *bean*. Para que um *bean* seja utilizado através de uma ferramenta gráfica, então, ele deve ser empacotado num arquivo do tipo JAR juntamente com todas as classes e arquivos que ele requer [Freire, 00].

A API *JavaBeans* faz parte do JDK1.1 e qualquer ferramenta compatível com ele suporta implicitamente os conceitos e características envolvidos.

Apêndice B

8. Componentes *Ad Hoc* 802.11

Aqui são mostrados os diagramas de classes que representam os componentes *ad hoc* 802.11. Estes diagramas mostram os relacionamentos mais importantes dos componentes abordados bem como as *interfaces* e classes abstratas que implementam.

O objeto quadro representa o quadro de dados que transita por uma rede *ad hoc* IEEE 802.11. Abaixo, na Figura 8.1, é apresentada a estrutura dessa classe.

Atributos:

- *idQuadro - Atributo do tipo int usado como identificador dessa classe.*
- *tempoGeracao – Atributo do tipo float que representa o tempo de criação do quadro.*
- *noCorrente – Atributo do tipo int que representa o identificador do elemento do modelo de rede em que se encontra o quadro.*
- *idElementoFonte - Atributo do tipo int usado como identificador da fonte que criou o quadro.*
- *rota – Atributo do tipo int usado como identificador da rota do quadro.*
- *tempodeFilaEstatisticas – Atributo do tipo AcumuladorEstatistico usado para coletar estatísticas sobre o tempo de fila do quadro no decorrer da simulação.*
- *distribuicaoProbabilidadeQuadro – Atributo do tipo int que indica o valor do tamanho do quadro de acordo ou não com uma distribuição de probabilidade. Se este atributo assumir um valor positivo então este valor será o tamanho do quadro, caso contrário, trata-se de uma distribuição cujo identificador é o próprio valor negativo.*
- *exponencialValorMedioQuadro – Atributo do tipo int que representa o valor médio do tamanho do quadro que será usado para calcular uma amostra da distribuição de probabilidade que representará o tamanho do quadro.*

- *unifomeBQuadro – Atributo do tipo int que representa o segundo valor da distribuição de probabilidade do tipo uniforme que será usada para calcular o tamanho do quadro.*
- *uniformeAQuadro – Atributo do tipo int que representa o primeiro valor da distribuição de probabilidade do tipo uniforme que será usada para calcular o tamanho do quadro.*
- *tempodeFilaEntrada – Atributo do tipo float que representa o tempo em que o quadro entrou na fila presente no elemento de modelagem.*

Métodos:

- *setIdQuadro()* e *getIdQuadro()* – Definem a propriedade *idQuadro*.
- *getTempoFilaEntrada()* e *setTempoFilaEntrada()* – Definem a propriedade *tempoFilaEntrada*.
- *getTempodeFilaEstatisticas()* e *setTempodeFilaEstatisticas()* – Definem a propriedade *tempodeFilaEstatisticas*.
- *getTempoGeracao()* e *setTempoGeracao()* – Definem a propriedade *tempoGeracao*.
- *getRota()* e *setRota()* – Definem a propriedade *rota*.
- *getIdElementoFonte()* e *setIdElementoFonte()* – Definem a propriedade *idElementoFonte*.
- *getNoCorrente()* e *setNoCorrente()* – Definem a propriedade *noCorrente*.
- *getDistribuicaoProbabilidadeQuadro()* e *setDistribuicaoProbabilidadeQuadro()* – Definem a propriedade *distribuicaoProbabilidadeQuadro*.
- *getExponencialValorMedioQuadro()* e *setExponencialValorMedioQuadro()* – Definem a propriedade *exponencialValorMedioQuadro*.
- *getUniformeAQuadro()* e *setUniformeAQuadro()* – Definem a propriedade *uniformeAQuadro*.
- *getUniformeBQuadro()* e *setUniformeBQuadro()* – Definem a propriedade *uniformeBQuadro*.
- *criaId()* - Método usado para criar o identificador do componente.

- *criaTempodeFilaEstatisticas()* – Método usado para instanciar o acumulador *tempodeFilaEstatisticas*.

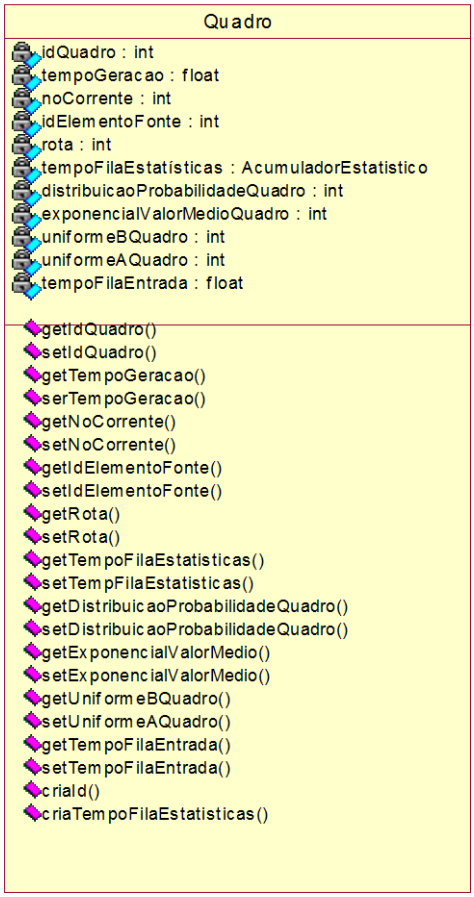

Figura 8.1 - Classe Quadro.

Em virtude do grande número de métodos, o componente *Fonte* é apresentado em duas Figuras (8.2 e 8.3). Na primeira, são descritos os atributos e os métodos das propriedades. Na segunda figura são descritos os demais métodos.

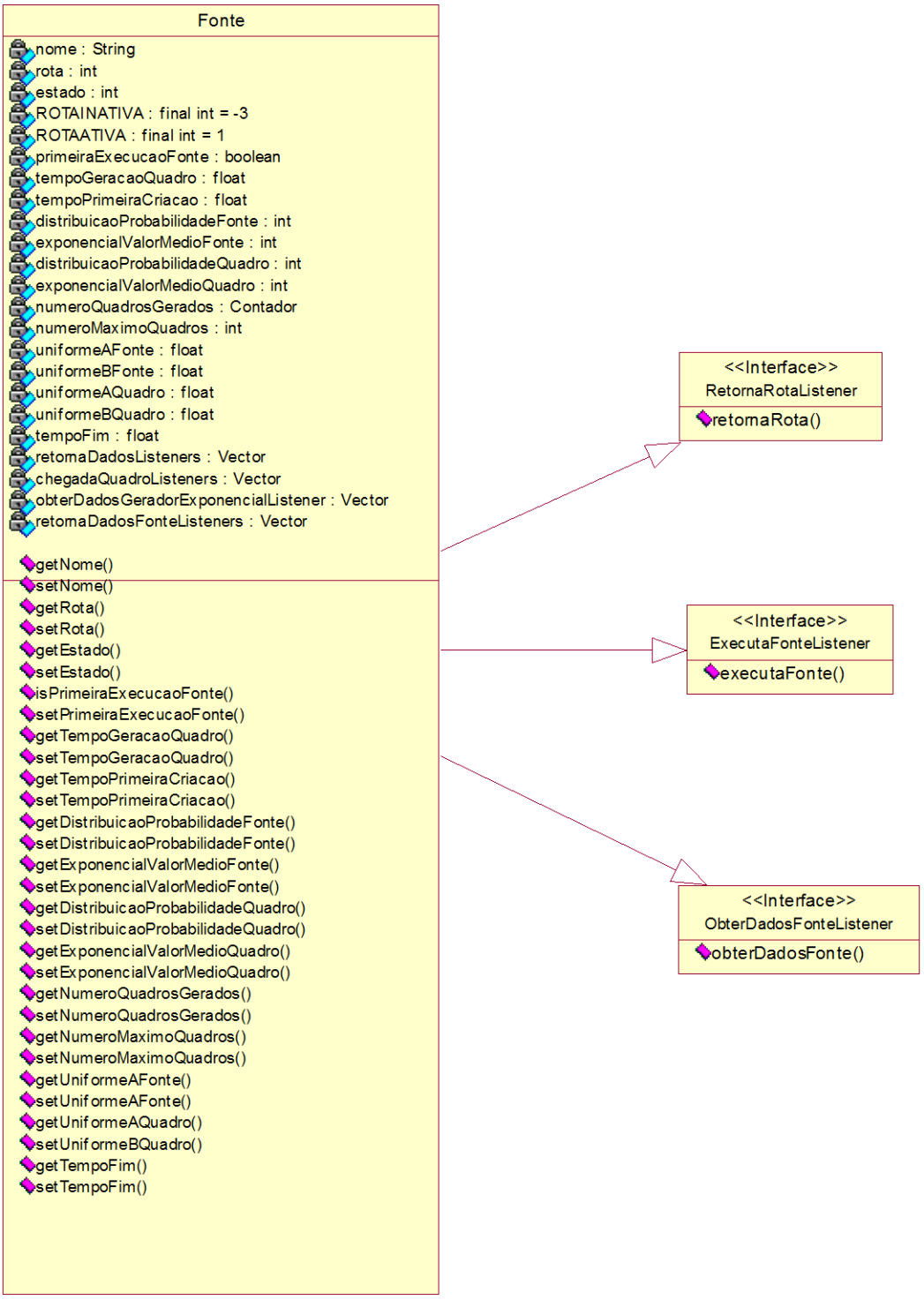

Figura 8.2 - Componente Fonte – Atributos e Propriedades.

Atributos:

- *nome – String em Java utilizado para armazenar o nome do componente.*
- *rota – Atributo do tipo int usado para identificar o número da rota.*

- *estado – Atributo do tipo int usado para guardar o número que corresponde ao estado do componente.*
- *ROTAINATIVA – Constante usada para atribuir um valor inteiro, no caso -3, ao estado da rota caso esta se encontre inativa.*
- *ROTAATIVA – Constante usada para atribuir um valor inteiro, no caso 1, ao estado da rota caso esta se encontre ativa.*
- *primeiraExecucaoFonte– Atributo do tipo boolean usado para identificar se a fonte já teve a sua primeira execução.*
- *tempoGeracaoQuadro – Atributo do tipo float usado para armazenar o tempo em que o quadro deve ser criado.*
- *tempoPrimeiraCriacao – Atributo do tipo float usado para armazenar o tempo em que a fonte deve criar o primeiro quadro.*
- *distribuiçãoProbabilidadeFonte – Atributo do tipo float usado para armazenar o valor da distribuição de probabilidade da fonte que foi inserido pelo usuário. Se for um valor negativo significa que a fonte precisa usar o gerador de números aleatórios a fim de conseguir uma amostra do tempo de interchegada que será usado na criação dos quadros. Caso contrário, se for um número positivo este valor será o utilizado como tempo de interchegada na criação dos quadros.*
- *exponencialValorMedioFonte – Atributo do tipo float usado para armazenar o valor médio que será utilizado pela função de distribuição de probabilidade.*
- *distribuiçãoProbabilidadeQuadro – Atributo do tipo int usado para armazenar o valor da distribuição de probabilidade para o tamanho do quadro que foi inserido pelo usuário. Se for um valor negativo significa que o tamanho do quadro precisa ser calculado através de um gerador de números aleatórios Caso contrário, se for um número positivo este valor será o utilizado para o tamanho do quadro.*
- *exponencialValorMedioQuadro – Atributo do tipo int usado para armazenar o valor médio que será utilizado pela função de distribuição de probabilidade.*
- *numeroQuadrosGerados – Atributo do tipo Contador usado para calcular o número de quadros gerados pela fonte.*

- *numeroMaximoQuadros – Atributo do tipo int usado para estabelecer o número máximo de quadros que devem ser gerados pela fonte.*
- *uniformeAFonte – Atributo do tipo float usado para determinar o valor do primeiro número que a distribuição uniforme utiliza para realizar os seus cálculos.*
- *uniformeBFonte – Atributo do tipo float usado para determinar o valor do segundo número que a distribuição uniforme utiliza para realizar os seus cálculos.*
- *uniformeAQuadro – Atributo do tipo float usado para determinar o valor do primeiro número que a distribuição uniforme utiliza para realizar os seus cálculos.*
- *uniformeBQuadro – Atributo do tipo float usado para determinar o valor do segundo número que a distribuição uniforme utiliza para realizar os seus cálculos.*
- *tempoFim – Atributo do tipo float usado para armazenar o tempo em que a simulação deve ser interrompida.*
- *retornaDadosListeners – Vector em Java usado para armazenar os listeners do evento RetornaDadosEvent.*
- *obterRotaListeners – Vector em Java usado para armazenar os listeners do evento ObterRotaEvent.*
- *chegadaQuadroListeners – Vector em Java usado para armazenar os listeners do evento ChegadaQuadroEvent.*
- *obterDadosGeradorExponencialListeners – Vector em Java usado para armazenar os listeners do evento obterDadosGeradorExponencialEvent.*
- *retornaDadosFonteListeners – Vector em Java usado para armazenar os listeners do evento retornaDadosFonteEvent.*

Métodos:

- *setNome() e getNome() – Definem a propriedade nome.*
- *setTempoFim() e getTempoFim() – Definem a propriedade tempoFim.*
- *setNumeroQuadrosGerados() e getNumeroQuadrosGerados() – Definem a propriedade numeroQuadrosGerados.*
- *setUniformeAQuadro() e getUniformeAQuadro() – Definem a propriedade uniformeAQuadro.*

- *setUniformeBQuadro() e getUniformeBQuadro() – Definem a propriedade uniformeBQuadro.*
- *setExponencialValorMedioQuadro() e getExponencialValorMedioQuadro() – Definem a propriedade exponencialValorMedioQuadro.*
- *setDistribuicaoProbabilidadeQuadro() e getDistribuicaoProbabilidadeQuadro() – Definem a propriedade distribuicaoProbabilidadeQuadro.*
- *setEstado() e getEstado() – Definem a propriedade estado.*
- *setTempoPrimeiraCriacao() e getTempoPrimeiraCriacao() – Definem a propriedade tempoPrimeiraCriacao.*
- *setExponencialValorMedioFonte() e getExponencialValorMedioFonte() – Definem a propriedade exponencialValorMedioFonte.*
- *setUniformeAFonte() e getUniformeAFonte() – Definem a propriedade uniformeAFonte.*
- *setUniformeBFonte() e getUniformeBFonte() – Definem a propriedade uniformeBFonte.*
- *setNumeroMaximoQuadros() e getNumeroMaximoQuadros() – Definem a propriedade numeroMaximoQuadros.*
- *setPrimeiraExecucaoFonte() e isPrimeiraExecucaoFonte() – Definem a propriedade primeiraExecucaoFonte.*
- *setDistribuicaoProbabilidadeFonte() e getDistribuicaoProbabilidadeFonte() – Definem a propriedade distribuicaoProbabilidadeFonte.*
- *setTempoGeracaoQuadro() e setTempoGeracaoQuadro() – Definem a propriedade tempoGeracaoQuadro.*
- *setRota() e getRota() – Definem a propriedade rota.*

A Figura 8.3 apresenta a segunda parte do componente *Fonte* enfocando os seus métodos. Não foram reapresentadas as *interfaces* da Figura 8.2.

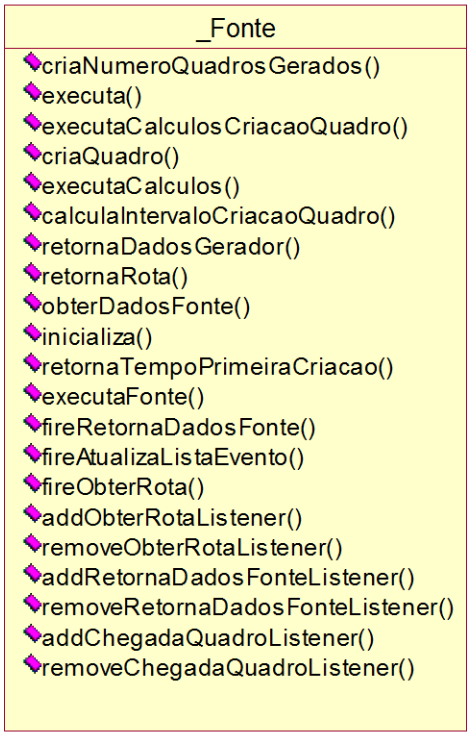

Figura 8.3 - Componente Fonte – Métodos.

Métodos:

- *criaNumeroQuadrosGerados() – Método utilizado para criar uma instância do contador numeroQuadrosGerados.*
- *executa() – Método responsável pela execução da fonte chamado quando ocorre o evento ExecutaFonteEvent.*
- *executaCalculosCriacaoQuadro() Método responsável em criar um objeto da classe Quadro.*
- *criaQuadro() – Método responsável em criar um quadro no caso do seu tamanho for de acordo com um distribuição de probabilidade.*
- *executaCalculos() – Método responsável pela execução da fonte chamado dentro do método executa().*
- *calculaIntervaloCriacaoQuadro() – Método responsável em calcular o intervalo que o componente Fonte deve usar para criar os quadro do modelo.*

- *retornaDadosGerador() - Método da interface retornaDadosListener usado para retornar o valor obtido da distribuição de probabilidade.*
- *retornaRota() - Método da interface retornaRotaListener usado para retornar a rota que a fonte deve usar para criar os quadros.*
- *obterDadosFonte() - Método da interface ObterDadosFonteListener usado para retornar os dados coletados pelas fontes para o ProcessadorMedidasDesempenho.*
- *inicializa() - Método da interface IncializaListener que realiza o processo de inicialização do componente. No caso da fonte ocorre também a inicialização de seu contador interno.*
- *retornaTempoPrimeiraCriacao() – Método usado para decidir qual é o tempo em que o primeiro quadro deve ser gerado.*
- *executaFonte() - Método da interface ExecutaFonteListener que faz com que a fonte seja executada.*
- *fireRetornaDadosFonte() – Método usado para disparar o evento RetornaDadosFonteEvent.*
- *fireAtualizaListaEvento() – Método usado para disparar o evento ChegadaQuadroEvent.*
- *fireObterRota() – Método usado para disparar o evento ObterRotaEvent.*
- *addObterRotaListener() e removeObterRotaListener() - Métodos que fornecem o cadastro/descadastro de listeners para o evento ObterRotaEvent.*
- *addRetornaDadosFonteListener() e removeRetornaDadosFonteListener() - Métodos que fornecem o cadastro/descadastro de listeners para o evento RetornaDadosFonteEvent.*
- *addChegadaQuadroListener() e removeChegadaQuadroListener() - Métodos que fornecem o cadastro/descadastro de listeners para o evento ChegadaQuadroEvent*

Pelos mesmos motivos apresentados pelo componente *Fonte,* o componente *Host* é mostrado em duas Figuras (8.4 e 8.5). A primeira Figura apresenta os atributos e os métodos que definem as propriedades. A segunda Figura apresenta os demais métodos.

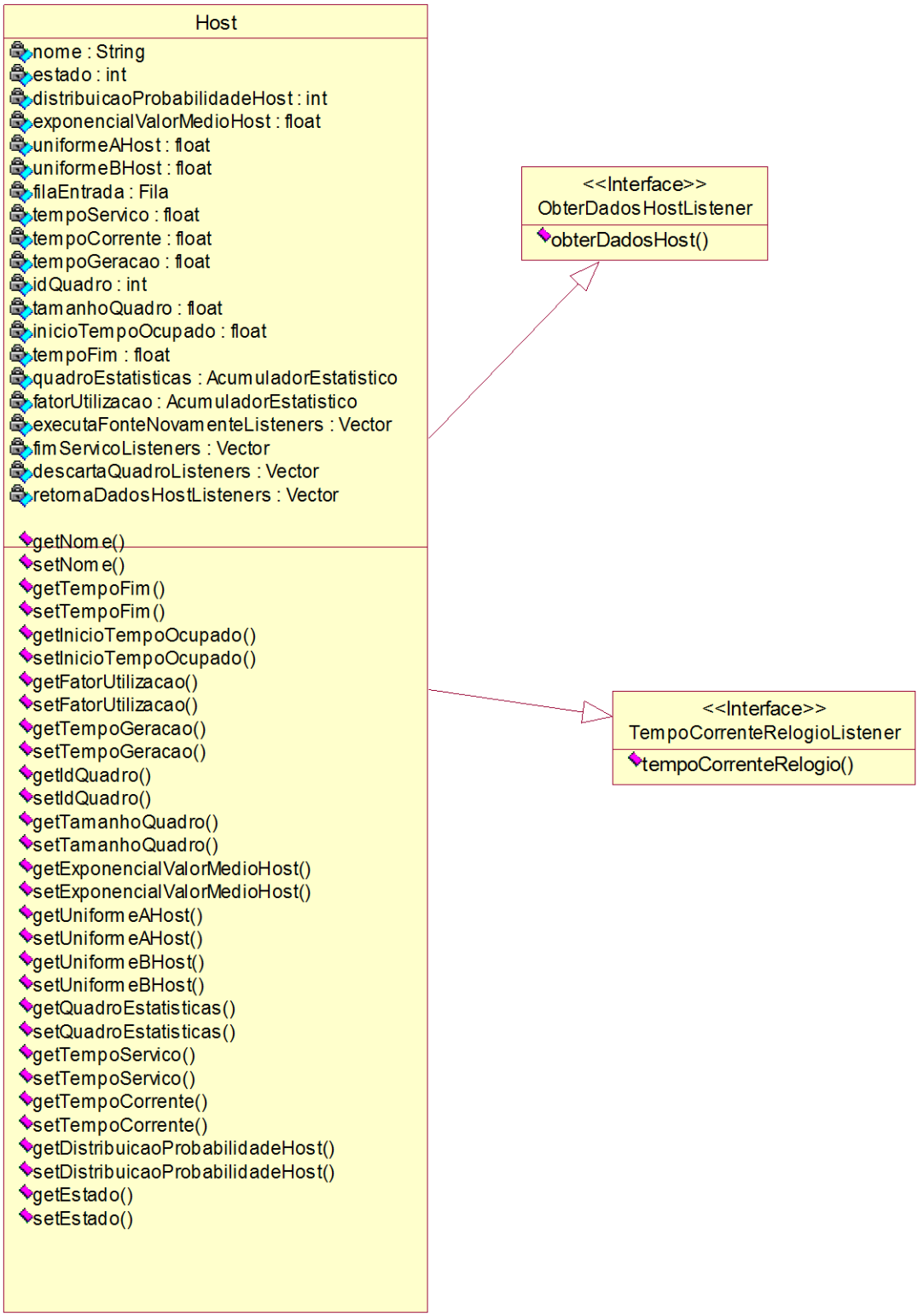

Figura 8.4 - Componente *Host*– Atributos e propriedades.

Atributos:

- *nome – String em Java utilizado para armazenar o nome do componente.*

- *estado – Atributo do tipo int usado para guardar o número que corresponde ao estado do componente.*
- *distribuiçãoProbabilidadeHost – Atributo do tipo float usado para armazenar o valor da distribuição de probabilidade do componente Host que foi inserido pelo usuário. Se for um valor negativo significa que o componente Host precisa usar um gerador de números aleatórios a fim de conseguir uma amostra do tempo de interchegada que será usado para determinar o tempo que um quadro leva para ser processado. Caso contrário, se for um número positivo, este valor será o utilizado para determinar o tempo de serviço.*
- *uniformeAHost– Atributo do tipo float usado para determinar o valor do primeiro número que a distribuição uniforme utiliza para realizar os seus cálculos.*
- *uniformeBHost – Atributo do tipo float usado para determinar o valor do segundo número que a distribuição uniforme utiliza para realizar os seus cálculos.*
- *filaEntrada – Atributo do tipo Fila que representa a fila de entrada do componente Host.*
- *tempoServico – Atributo do tipo float usado para armazenar o tempo que o Host leva para processar o quadro.*
- *tempoCorrente – Atributo do tipo float usado para armazenar o tempo corrente da simulação.*
- *tempoGeracao – Atributo do tipo float usado para armazenar o tempo em que o quadro foi transmitido. Ele corresponde a soma dos atributos tempoCorrente mais tempoServico.*
- *idQuadro – Atributo do tipo int usado como identificador do quadro.*
- *tamanhoQuadro – Atributo do tipo float usado para armazenar o tamanho do quadro.*
- *inicioTempoOcupado – Atributo do tipo float usado para determinar o tempo em que o componente começou a processar o quadro.*
- *tempoFim – Atributo do tipo float usado para armazenar o tempo em que a simulação deve ser interrompida.*

- *quadroEstatisticas – Atributo do tipo AcumuladorEstatistico usado para coletar as estatísticas relacionadas ao quadro. Nesse caso, esse acumulador coletar informações sobre o tamanho de cada quadro processado pelo componente Host.*
- *fatorUtilização – Atributo do tipo AcumuladorEstatistico usado para coletar informações referentes a utilização do componente.*
- *executaFonteNovamenteListeners – Vector em Java usado para armazenar os listeners do evento ExecutaFonteNovamenteEvent.*
- *fimServicoListeners – Vector em Java usado para armazenar os listeners do evento FimServicoEvent.*
- *descartaQuadroListeners – Vector em Java usado para armazenar os listeners do evento DescartaQuadroEvent.*
- *retornaDadosHostListeners – Vector em Java usado para armazenar os listeners do RetornaDadosHostEvent.*

Métodos:

- *setNome() e getNome() – Definem a propriedade nome.*
- *setTempoFim() e getTempoFim() – Definem a propriedade tempoFim.*
- *setInicioTempoOcupado() e getInicioTempoOcupado() – Definem a propriedade inicioTempoOcupado.*
- *setfatorUtilizacao() e getfatorUtilizacao() – Definem a propriedade fatorUtilizacao.*
- *setTempoGeracao() e getTempoGeracao() – Definem a propriedade tempoGeracao.*
- *setIdQuadro() e getIdQuadro() – Definem a propriedade idQuadro.*
- *setTamanhoQuadro() e getTamanhoQuadro() – Definem a propriedade tamanhoQuadro.*
- *setExponencialValorMedioHost() e getExponencialValorMedioHost() – Definem a propriedade exponencialValorMedioHost.*
- *setDistribuicaoProbabilidadeHost() e getDistribuicaoProbabilidadeHost() – Definem a propriedade distribuicaoProbabilidadeHost.*
- *setEstado() e getEstado() – Definem a propriedade estado.*

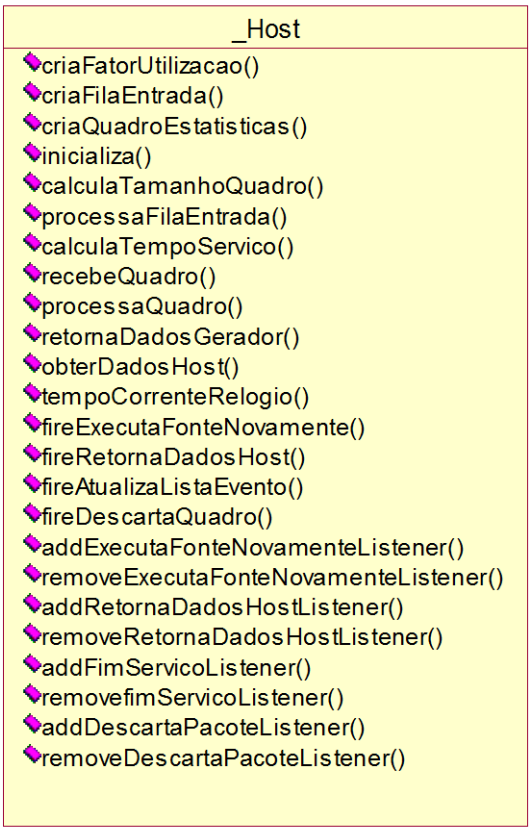

Figura 8.5 - Componente *Host*– Restante dos métodos.

Métodos:

- *criaFatorUtilizacao() – Método utilizado para criar uma instância do acumulador fatorUtilizacao.*
- *criaFilaEntrada() - Método utilizado para criar uma instância da classe Fila.*
- *criaQuadroEstatisticas() - Método utilizado para criar uma instância do acumulador quadroEstatisticas.*
- *inicializa() - Método da interface InicializaListener que realiza o processo de inicialização do componente. No caso do componente Host ocorre também a inicialização de suas filas e de seus acumuladores.*
- *calculaTamanhoQuadro() – Método usado para calcular o tamanho do quadro no caso do mesmo apresentar uma distribuição de probabilidade para o seu tamanho.*
- *processaFilaEntrada() – Método usado para processar os quadros que se encontram na fila de entrada do componente.*

- *calculaTempoServico() – Método usado para calcular quanto tempo o componente levará para processar o quadro.*
- *recebeQuadro() - Método da interface recebeQuadroListener usado para que o componente Host receba o quadro.*
- *processaQuadro() – Método responsável em processar o quadro passado como parâmetro.*
- *retornaDadosGerador() - Método da interface retornaDadosListener usado para retornar o valor obtido da distribuição de probabilidade.*
- *obterDadosHost() - Método da interface ObterDadosHostListener usado para retornar os dados coletados pelo componente Hoss para o componente ProcessadorMedidasDesempenho.*
- *tempoCorrenteRelogio() - Método da interface TempoCorrenteRelogioListener usado para atualizar o tempo corrente do relógio.*
- *fireExecutaFonteNovamente() – Método usado para disparar o evento ExecutaFonteNovamenteEvent.*
- *fireRetornaDadosHost() – Método usado para disparar o evento RetornaDadosHostEvent.*
- *fireDescartaQuadro() – Método usado para disparar o evento DescartaQuadroEvent.*
- *addExecutaFonteNovamente() e removeExecutaFonteNovamente() - Métodos que fornecem o cadastro/descadastro de listeners para o evento ExecutaFonteNovamenteEvent.*
- *addRetornaDadosHostListener() e removeRetornaDadosHostListener() - Métodos que fornecem o cadastro/descadastro de listeners para o evento RetornaDadosHostEvent.*
- *addFimServicoListener() e removeFimServicoListener() - Métodos que fornecem o cadastro/descadastro de listeners para o evento FimServicoEvent.*
- *addDescartaQuadroListener() e removeDescartaQuadroListener() - Métodos que fornecem o cadastro/descadastro de listeners para o evento DescartaQuadroEvent*.

A Figura 8.6 apresenta a estrutura do componente *Enlace*. A seguir são descritos os seus atributos e métodos.

Atributos:

- *nome – String em Java utilizado para armazenar o nome do componente.*
- *estado – Atributo do tipo int usado para guardar o número que corresponde ao estado do componente.*
- *capacidade – Atributo do tipo int usado para determinar a capacidade do enlace.*
- *filaSaida – Atributo do tipo Fila que representa a fila de saída do componente Enlace. Em um sistema real ela é a fila de saída do Roteador ou a fila de saída do Host.*
- *tempoTransmissao – Atributo do tipo float usado para armazenar o tempo que o Enlace leva para transmitir o quadro.*
- *tempoCorrente – Atributo do tipo float usado para armazenar o tempo corrente da simulação.*
- *tempoGeracao – Atributo do tipo float usado para armazenar o tempo em que o quadro foi transmitido. Ele corresponde a soma dos atributos tempoCorrente mais tempoTransmissao.*
- *tamanhoQuadro – Atributo do tipo float usado para armazenar o tamanho do quadro.*
- *TODOS – Constante do tipo int usada para determinar se todos os componentes do tipo Enlace devem retornar os seus valores.*
- *tempoTransmissaoEstatisticas – Atributo do tipo AcumuladorEstatistico usado para coletar as estatísticas relacionadas à transmissão dos quadros.*
- *fimServicoListeners – Vector em Java usado para armazenar os listeners do evento FimServicoEvent.*
- *descartaQuadroListeners – Vector em Java usado para armazenar os listeners do evento DescartaQuadroEvent.*
- *retornaDadosEnlaceListeners – Vector em Java usado para armazenar os listeners do RetornaDadosEnlaceEvent.*

- *chegadaQuadroListeners – Vector em Java usado para armazenar os listeners do ChegadaQuadroEvent.*

Métodos:

- *setNome() e getNome() – Definem a propriedade nome.*
- *setTempoFim() e getTempoFim() – Definem a propriedade tempoFim.*
- *setTempoGeracao() e getTempoGeracao() – Definem a propriedade tempoGeracao.*
- *setTempoTransmissaoEstatisticas() e getTempoTransmissaoEstatisticas() – Definem a propriedade tempoTransmissaoEstatisticas.*
- *setTamanhoQuadro() e getTamanhoQuadro() – Definem a propriedade tamanhoQuadro.*
- *setCapacidade() e getCapacidade() – Definem a propriedade capacidade.*
- *setFilaSaida() e getFilaSaida() – Definem a propriedade filaSaida.*
- *setTempoTransmissao() e getTempoTransmissao() – Definem a propriedade tempoTransmissao.*
- *setTempoCorrente() e getTempoCorrente() – Definem a propriedade tempoCorrente.*
- *setEstado() e getEstado() – Definem a propriedade estado.*
- *criaTempoTransmissaoEstatisticas() – Método utilizado para criar uma instância do acumulador tempoTransmissaoEstatisticas.*
- *criaFilaSaida() - Método utilizado para criar uma instância da classe Fila.*
- *processaFilaEntrada() – Método usado para processar os quadros que se encontram na fila de saida do componente.*
- *calculaTempoServico() – Método usado para calcular quanto tempo o componente levará para transmitir o quadro pelo Enlace.*
- *calculaTamanhoQuadro() – Método usado para calcular o tamanho do quadro no caso do mesmo apresentar uma distribuição de probabilidade para o seu tamanho.*
- *inicializa() - Método da interface InicializaListener que realiza o processo de inicialização do componente. No caso do componente Enlace ocorre também a inicialização de sua fila e de seus acumuladores.*

- *transmite() - Método da interface TransmiteListener usado para que o Enlace receba o quadro.*
- *transmiteQuadro() – Método responsável em transmitir o quadro passado como parâmetro.*
- *retornaDadosGerador() - Método da interface retornaDadosListener usado para retornar o valor obtido da distribuição de probabilidade.*
- *obterDadosEnlace() - Método da interface ObterDadosEnlaceListener usado para retornar os dados coletados pelo componente Enlace para o ProcessadorMedidasDesempenho.*
- *tempoCorrenteRelogio() - Método da interface TempoCorrenteRelogioListener usado para atualizar o tempo corrente do relógio.*
- *fireRetornaDadosEnlace() – Método usado para disparar o evento RetornaDadosEnlaceEvent.*
- *fireDescartaQuadro() – Método usado para disparar o evento DescartaQuadroEvent.*
- *fireAtualizaListaEvento() – Método usado para disparar o evento FimServicoEvent.*
- *addRetornaDadosEnlaceListener() e removeRetornaDadosEnlaceListener() - Métodos que fornecem o cadastro/descadastro de listeners para o evento RetornaDadosEnlaceEvent.*
- *addChegadaQuadroListener() e removeChegadaQuadroListener() - Métodos que fornecem o cadastro/descadastro de listeners para o evento ChegadaQuadroEvent.*
- *addDescartaQuadroListener() e removeDescartaQuadroListener() - Métodos que fornecem o cadastro/descadastro de listeners para o evento FimServicoEvent.*

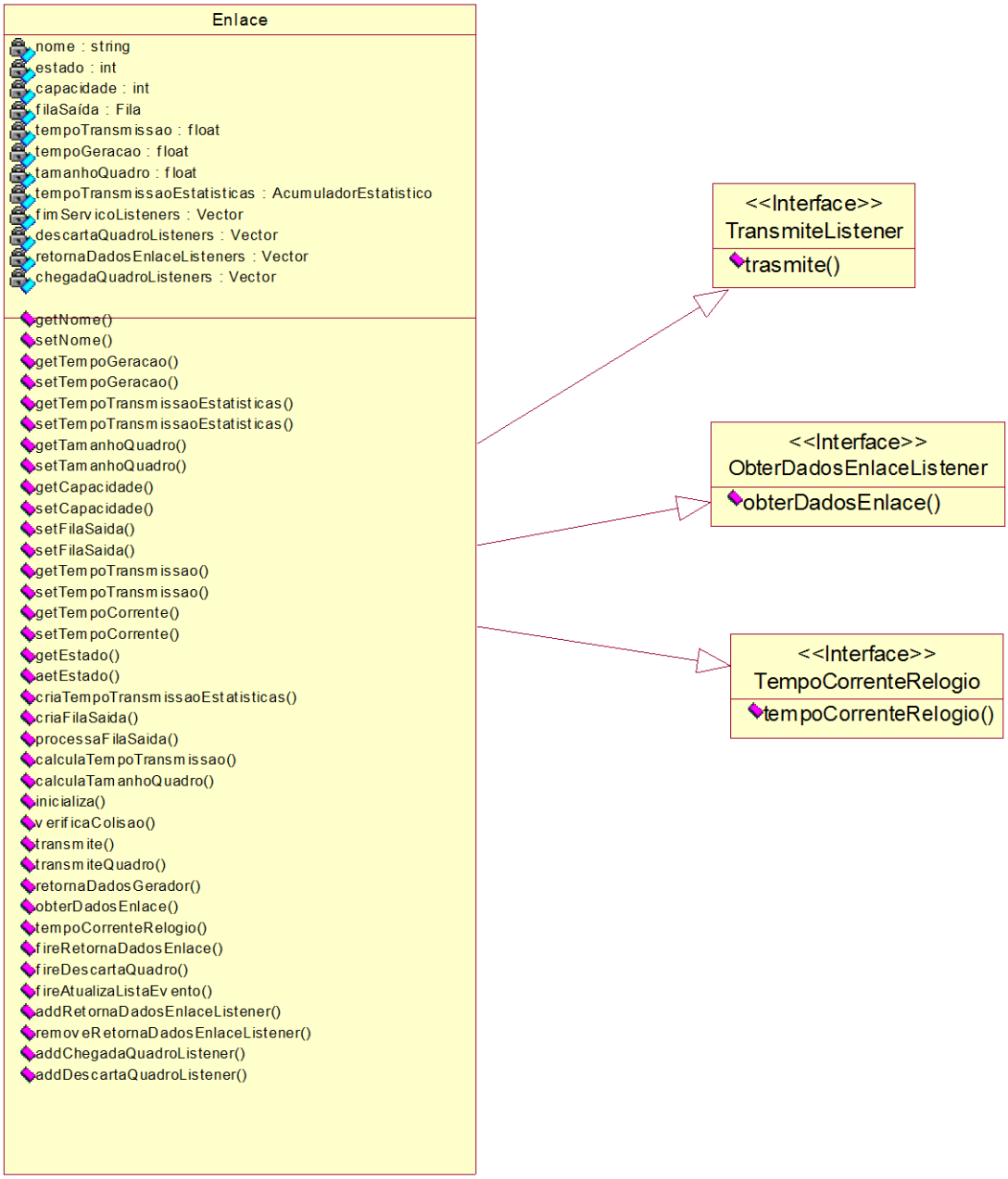

Figura 8.6 - Componente Enlace.

A Figura 8.7 apresenta a estrutura do último elemento de modelagem, o *Sorvedouro*.

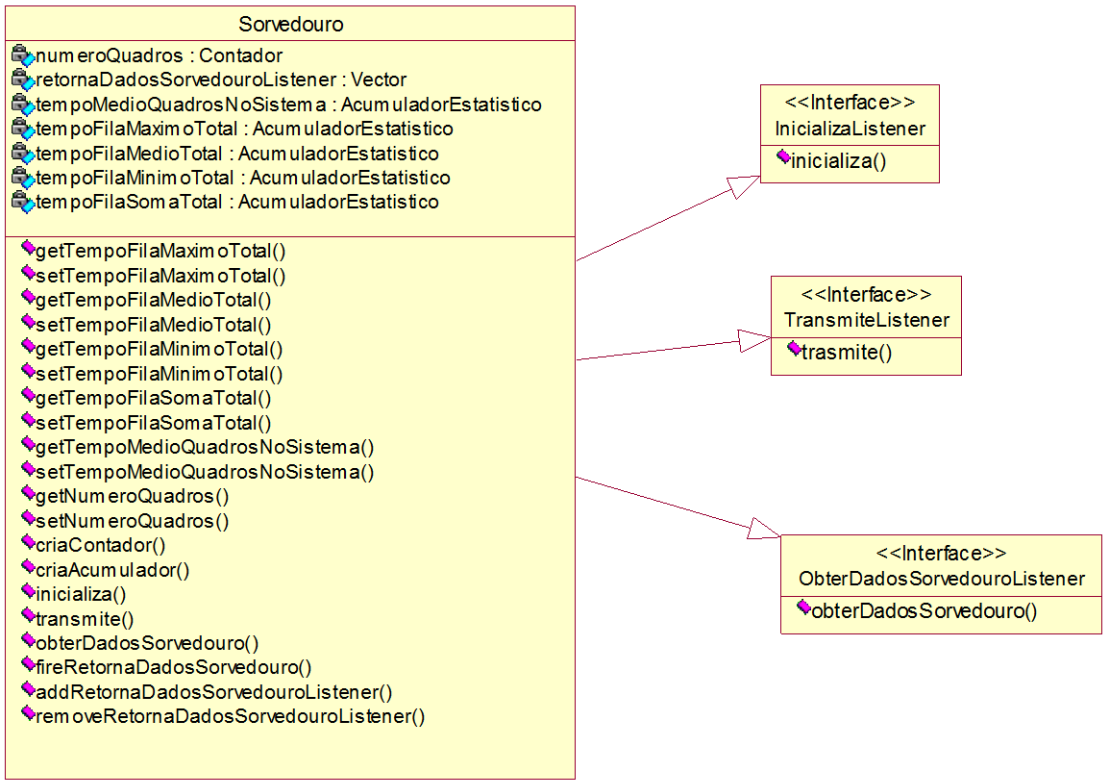

Figura 8.7 - Componente Sorvedouro.

Atributos:

- *numeroQuadros – Atributo do tipo Contador que armazena o número de quadros descartados.*
- *tempoMedioQuadrosNoSistema – Atributo do tipo AcumuladorEstatistico que armazena os tempos de todos os quadros do sistema.*
- *tempoFilaMinimoTotal – Atributo do tipo AcumuladorEstatistico que armazena os tempos de fila mínimos de todos os quadros.*
- *tempoFilaMédiaTotal – Atributo do tipo AcumuladorEstatistico que armazena os tempos de fila médios de todos os quadros.*
- *tempoFilaMaximoTotal – Atributo do tipo AcumuladorEstatistico que armazena os tempos de fila máximos de todos os quadros.*
- *tempoFilaSomaTotal – Atributo do tipo AcumuladorEstatistico que armazena todas as somas referentes aos tempos de fila de todos os quadros.*

- *retornaDadosSorvedouroListeners – Vector em Java usado para armazenar os listeners do RetornaDadosEnlaceEvent.*

Métodos:

- *SetTempoFilaMaximoTotal() e getTempoFilaMaximoTotal() – Definem a propriedade tempoFilaMaximoTotal.*
- *SetTempoFilaMinimoTotal() e getTempoFilaMinimoTotal() – Definem a propriedade tempoFilaMinimoTotal.*
- *SetTempoFilaMediaTotal() e getTempoFilaMediaTotal() – Definem a propriedade tempoFilaMediaTotal.*
- *SetTempoFilaSomaTotal() e getTempoFilaSomaTotal() – Definem a propriedade tempoFilaSomaTotal.*
- *SetTempoMedioQuadrosNoSistema() e getTempoMedioQuadrosNoSistema() – Definem a propriedade tempoMedioQuadrosNoSistema.*
- *setNumeroQuadros() e getNumeroQuadros() – Definem a propriedade numeroQuadros.*
- *criaContador() – Método utilizado para criar uma instância da classe Contador.*
- *criaAcumulador() – Método utilizado para criar uma instância da classe AcumuladorEstatístico.*
- *inicializa() - Método da interface InicializaListener que realiza o processo de inicialização do componente. No caso do componente Sorvedouro a inicialização consiste em inicializar os seus contadores e acumuladores.*
- *transmite() - Método da interface transmiteListener usado para que o Sorvedouro receba o quadro.*
- *obterDadosSorvedouro() - Método da interface ObterDadosSorvedouroListener usado para retornar os dados coletados pelos Sorvedouro para o ProcessadorMedidasDesempenho.*
- *fireRetornaDadosSorvedouro() – Método usado para disparar o evento RetornaDadosSorvedouroEvent.*