Universidade Federal de Pernambuco

Centro de Informática

Graduação em Ciência da Computação

TOSViz: Ferramenta de Visualização de Simulação para Aplicações de Redes de Sensores Sem Fio Baseadas no TinyOS

Recife – Dezembro de 2008.

Universidade Federal de Pernambuco

Centro de Informática

Graduação em Ciência da Computação

TOSViz: Ferramenta de Visualização de Simulação para Aplicações de Redes de Sensores Sem Fio Baseadas no TinyOS

Trabalho de Graduação

Rilter Tavares do Nascimento

Orientador: Nelson Souto Rosa

Este trabalho foi apresentado ao Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para a obtenção do Grau de Bacharel em Ciências da Computação.

Recife – Dezembro de 2008.

A minha família

Agradecimentos

Gostaria de agradecer primeiramente a Deus por permitir o sucesso da minha caminhada na vida até agora e a busca por novas oportunidades que contribuirão para o meu crescimento pessoal e profissional.

Agradeço ao professor e orientador Nelson Rosa. Pessoa exemplo de profissionalismo e dedicação a comunidade acadêmica, um verdadeiro mestre. Ele se disponibilizou e ajudou bastante na confecção deste trabalho.

Agradeço aos meus pais Railson e Marli pelo carinho, conselhos, paciência e dedicação ao longo de meu crescimento.

Agradeço ao único amor da minha vida, Acácia. Mulher maravilhosa que vem me ajudando a vencer os obstáculos da vida e compartilhando momentos únicos e inesquecíveis.

Agradeço a minha irmã mais velha, Riviane. Ser humano que amo, respeito e admiro quando não atrapalha minhas atividades para ver o *Orkut*.

Agradeço a Vivian, minha sobrinha de 1 ano e 3 meses, que nos momentos de preocupação no desenvolvimento deste trabalho, aparecia gritando "titio" e sentando no meu colo para bater em todas as teclas do teclado, puxar o fio do fone de ouvido e derrubar o *mouse* no chão.

Agradeço a família que vem me dando apoio na luta pelas minhas conquistas.

Agradeço aos amigos Buiu e Fundinho pelo companheirismo. Pessoas que conheço há muito tempo e que passaram este último período atrapalhando minhas horas de estudo para sair e se divertir com as namoradas.

Agradeço aos amigos Chico, David, Digao, Japa, Felipe Moxinho, Gravatá, Guedinho, Hélio, Henrique, Papel, Petrolina, Pigmeu, Tanga, Vivi e Xeroso B. pelos momentos de descontração dentro e fora do CIn e pelas caronas que peguei.

Resumo

Redes de sensores sem fio (RSSFs) têm sido utilizadas em diversas áreas que incluem serviços de segurança, ecologia, agricultura, saúde, militar, logística, robótica, e assim por diante. O TinyOS tem sido o principal sistema operacional para RSSFs e o TOSSIM o simulador de RSSFs mais utilizado. Neste contexto, o objetivo deste trabalho é implementar uma ferramenta de visualização de simulações TOSSIM para aplicações de RSSFs desenvolvidas no TinyOS 2.x. A criação desta ferramenta é de grande utilidade para a comunidade que desenvolve aplicações para RSSF, pois permite a visualização e análise de dados da simulação da aplicação antes de sua implementação no ambiente físico real.

Palavras-Chaves: sensores, simulador, sistemas distribuídos, tinyos, tossim.

Abstract

Wireless sensor networks (WSNs) have been used in many areas such as security services, ecology, agriculture, health, military, logistic, robotic and so on. The TinyOS has been the principal operating system for WSNs and the TOSSIM is the most used simulator. In this context, the aim of this work is developing a visualization tool of TOSSIM simulations for WSNs applications developed in TinyOS 2.x. The creation of this tool is very interested by the community which develops WSNs applications because it allows the visualization and the analysis when simulating applications before theirs deployment in a real scenario.

Keywords: *sensors, simulator, distributed systems, tinyos, tossim.*

Sumário

Índice de Imagens

Índice de Tabelas

"Tu te tornas eternamente responsável por aquilo que cativas."

(Pequeno Príncipe, Antoine de Saint-Exupéry)

1 Introdução

O uso de sensores facilita o monitoramento de ambientes físicos. Desta forma, redes de sensores podem ser utilizadas em diversos domínios de aplicação [3][6], tais como: serviços de segurança, ecologia, agricultura, saúde, militar, logística, robótica, e assim por diante.

Em aplicações militares, o interesse vai desde a coleta de dados até o rastreamento do inimigo ou sobrevivência no campo de batalha. Um cenário simples de entender é a utilização de um sistema de defesa que através de sensores substitui minas, consideradas perigosas e obsoletas, para detecção de invasores em territórios. Em outro cenário, agora para ambientes internos, duas redes de sensores podem integrar a detecção de fumaça e incêndio e os sinais luminosos que indicam saídas. Em caso de emergência, a RSSF é responsável por encaminhar as pessoas para a saída através do caminho mais seguro. Por fim, na agricultura, a irrigação pode ser controlada de forma mais econômica e eficiente ao monitorar o solo, a umidade do ar e clima.

1.1 Contexto

Os sensores podem estar dispostos em uma rede com ou sem fio. As Redes de Sensores Sem Fio (RSSFs) vêm sendo amplamente utilizadas em conseqüência do desenvolvimento da tecnologia de comunicação sem fio e apresentam vantagens sobre as redes com fio (e.g., facilidade de instalação, mobilidade dos nós). As RSSFs possuem as seguintes características [3]: topologia dinâmica, grande quantidade de sensores, sensores são propensos a falhas, comunicação *broadcast*, sensores possuem limitação de energia, capacidade computacional e memória, e ausência de identificadores globais por causa da grande quantidade de sensores.

A Figura 1.1 mostra como os sensores geralmente são espalhados em uma área que está sendo monitorada. Cada sensor tem a capacidade de coletar informações e roteá-las ao nó *Sink* que é o responsável por se comunicar com a aplicação para gerenciamento de sensores através da Internet.

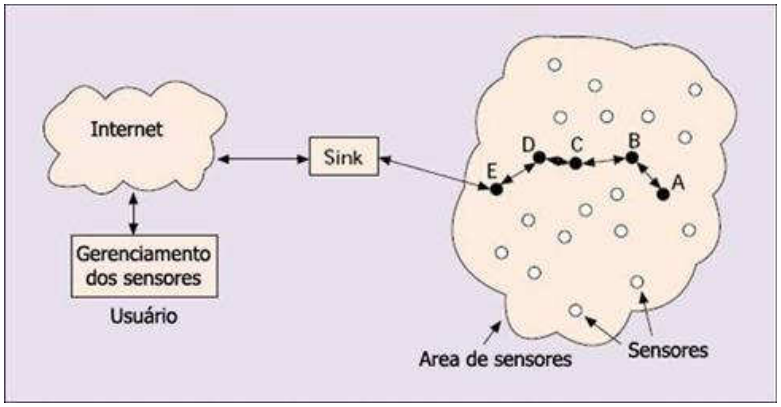

Figura 1.1: Elementos de uma RSSF [3]

Muitos sensores funcionam com o TinyOS [14][23], um sistema operacional especialmente projetado para RSSF. Sua arquitetura é baseada em componentes que permitem rápida implementação, enquanto minimiza o tamanho do código respeitando a restrição de memória dos sensores. A biblioteca de componentes inclui protocolos de rede, serviços distribuídos, *drivers* de sensores e ferramentas de aquisição de dados.

O TinyOS é utilizado no desenvolvimento e teste de vários algoritmos e protocolos. Para simulações, ele é acompanhado pela biblioteca TOSSIM [25] que suporta duas interfaces de programação: C++ e Python. O desenvolvedor escreve um *script* de simulação em uma dessas linguagens que configura e executa a aplicação desenvolvida. O TOSSIM possui um sistema de *output* de depuração. Geralmente, o programador utiliza o console como saída para as informações da depuração. Um exemplo de *output* gerado pelo TOSSIM está mostrado na Figura 1.2.

```
DEBUG (1): Received active message (0x809b7c9) of type 112 and
           length 13 for me @ 0:0:9.808303807.
DEBUG (1): Received upper packet. Will signal up
DEBUG (1): LI receiving packet, buf addr: 809b7d0
2 11 80 ff ff 0 0 0 1 56 0 3 66
DEBUG (1): Got seq: 17 from link: 2
DEBUG (1): Found the entry so updating
DEBUG (1): updateNeighborEntryIdx: prevseq 16, curseq 17, gap 1
DEBUG (1): LinkEstimatorP$updateNeighborTableEst
DEBUG (1): Making link: 1 mature
DEBUG (1): MinPkt: 3, totalPkt: 3
DEBUG (3): newlen1 = 5
DEBUG (3): Max payload is: 28, maxEntries is: 7
DEBUG (3): Loaded on footer: 0 1 86
DEBUG (3): Loaded on footer: 1 2 117
DEBUG (3): newlen2 = 13
DEBUG (3): Sending seq: 17
2 10 80 ff ff 0 0 0 1 56 0 2 75
DEBUG (3): AM: Sending packet (id=112, len=13) to 65535
DEBUG (2): Received active message (0x809b7f2) of type 112 and
           length 13 for me @ 0:0:9.888351449.
DEBUG (2): Received upper packet. Will signal up
DEBUG (2): LI receiving packet, buf addr: 809b7f9
```

Figura 1.2: Exemplo de saída gerada pelo TOSSIM

Além do TOSSIM, existem outras ferramentas de simulação para aplicações de RSSF [1], tais como: NS-2 [16], Mannasim [15], SENSE [22], OMNet++ [17], Castalia [9], Qualnet [21], VisualSense [27], AlgoSenSim [4] e GTNetS [12].

1.2 Objetivo

A Versão 1.x do TinyOS possui uma ferramenta de visualização, o TinyViz. Na sua versão atual, Versão 2.x, o TinyViz não foi adaptado e deixou de ser distribuído junto com o TinyOS.

O objetivo deste trabalho é implementar uma ferramenta de visualização de simulação para aplicações de RSSF desenvolvidas no TinyOS 2.x. Neste caso, mensagens de simulação geradas pelo TOSSIM são recebidas e interpretadas pela ferramenta que as apresenta de forma visual.

A criação dessa ferramenta de visualização é de grande utilidade para a comunidade que desenvolve aplicações para RSSF. Com o TOSSIM não há a necessidade de realizar o *deploy* em vários sensores a cada rodada de testes e validações diminuindo o *overhead* dessa fase de desenvolvimento. Porém, a simulação textual dificulta a observação e extração de informações dos eventos gerados. O TOSViz agrega ao simulador TOSSIM funcionalidades que permitem melhor clareza sobre os eventos, apresentação de dados estatísticos do uso de recursos, visualização dos *traces* da simulação, visualização e manipulação da topologia e representação gráfica do comportamento da rede.

1.3 Organização do documento

Além deste capítulo introdutório, este documento está organizado da seguinte forma:

Capítulo 2: apresenta conceitos básicos para o entendimento da solução proposta.

Capítulo 3: contém informações de outras ferramentas para visualização de simulação de aplicações para RSSF.

Capítulo 4: mostra as principais funcionalidades, arquitetura, projeto e implementação da ferramenta desenvolvida ao longo deste trabalho.

Capítulo 5: contém a avaliação da ferramenta proposta.

Capítulo 6: apresenta as conclusões obtidas durante o desenvolvimento deste trabalho, dificuldades encontradas e propostas de trabalhos futuros.

2 Conceitos Básicos

Neste capítulo são apresentados os conceitos básicos para entendimento da solução proposta neste trabalho. Inicialmente são apresentados os elementos básicos de uma Rede de Sensores Sem Fio. Em seguida, são apresentados o TinyOS e a linguagem nesC, utilizada para implementação de aplicações. Por fim, é apresentada a ferramenta de simulação do TinyOS, o TOSSIM.

2.1 Redes de Sensores Sem Fio

Os sensores sem fio são pequenos dispositivos capazes de medir, coletar e armazenar algum tipo de grandeza física no ambiente que o rodeia. Estas informações também podem ser enviadas para outros sensores. Desta forma, quando existem vários sensores, dispostos em uma área de monitoração, se comunicando e trocando mensagens entre si, há o que se define de Redes de Sensores Sem Fio (RSSF) e onde cada sensor consiste de um nó da rede. Um nó da RSSF, geralmente chamado de *Sink*, é o responsável por comunicar-se com a aplicação de gerenciamento dos sensores e de exibição e manipulação dos dados coletados para o usuário.

No momento em que uma RSSF é criada, alguns fatores devem ser levados em consideração. Akyildiz [3] define que fatores são importantes, pois servem como guia na criação de um protocolo ou um algoritmo para redes de sensores:

- **Tolerância a falhas**: alguns nós podem falhar por motivo de falta de energia, danos físicos ou interferência ambiental. Falhas de sensores não devem afetar, de forma geral, o funcionamento da rede;
- **Escalabilidade**: a quantidade de sensores monitorando um ambiente pode chegar à ordem de milhões dependendo da aplicação. As RSSFs precisam suportar essa capacidade bem como uma alta densidade de nós;
- **Custos da produção**: sabendo que as RSSFs possuem uma grande quantidade de nós, é importante que cada sensor tenha um valor baixo, pois o uso de uma rede sem fio não é economicamente justificado se o seu custo for muito superior ao de sensores tradicionais;

- **Restrições de hardware**: devido à necessidade de serem pequenos e leves, para facilitar a instalação em locais com diferentes peculiaridades, os sensores sem fio têm baixa capacidade computacional e pouca memória. Eles devem ter baixo consumo de energia visando evitar o incômodo da troca constante de bateria das centenas de sensores dispostos na rede;
- **Topologia**: quando uma rede possui centenas de nós, sua manutenção se torna mais difícil. Na fase de instalação, é preciso definir se os sensores vão ser colocados um a um por pessoas ou robô, ou em massa (e.g., por mísseis). Na fase de pós-instalação, deve-se gerenciar a energia disponível e os sensores que deixam de funcionar ou mudam de posição. Na fase de reinstalação, alguns nós precisam ser adicionados ou substituídos por causa de mau funcionamento ou mudança atividade;
- **Ambiente**: os sensores precisam funcionar em lugares remotos críticos como, por exemplo, no fundo do oceano, numa área biológica ou quimicamente contaminada, num campo de batalha ou em grandes construções;
- **Meio de transmissão**: a comunicação é feita através de um meio sem fio utilizando ondas de rádio, infravermelho ou *Bluetooth* por exemplo. A escolha do uso de um meio deve-se levar em consideração vários pontos desde viabilidade econômica a tolerância a falhas; e
- **Consumo de energia**: os sensores possuem limitadas fontes de energia. Existem casos em que não é possível realizar a troca de baterias.

2.2 TinyOS

O TinyOS [14][23] é um sistema operacional desenvolvido pela Universidade da Califórnia que possui uma arquitetura baseada em componentes e foi especialmente projetado para redes de sensores. As suas aplicações são escritas em um dialeto de C, denominado nesC (*network embedded systems C*) [11][14], que foi desenvolvido com o objetivo de ser eficiente em relação ao limite de memória dos sensores e permitir rápida implementação.

Um programa do TinyOS pode ser visto como um grafo de componentes os quais possuem variáveis privadas que podem ser referenciadas apenas pelo próprio componente. Os componentes possuem três abstrações computacionais:

- *Commands*: requisição ao componente para execução de algum serviço como, por exemplo, enviar um pacote para outros sensores;
- *Events*: sinalização do componente indicando o fim da execução de um serviço. A separação da requisição (*command*) e sinalização do final do serviço (*event*), chamada de *split-phase*, é conseqüência da estratégia do TinyOS de não bloquear durante a execução do serviço;
- *Tasks*: função executada pelo escalonador do TinyOS em algum momento posterior. Isto permite que *commands* e *events* tenham respostas imediatas ao passar a responsabilidade de tarefas, consideradas com maior complexidade computacional, para *tasks*. Enquanto *commands* e *events* são mecanismos para comunicação entre componentes, *tasks* representam concorrência interna no componente. O escalonador de *tasks* do TinyOS utiliza uma política de escalonamento FIFO não preemptiva.

Os componentes disponibilizam seus *commands* e *events* através de interfaces que podem ser usadas por outros componentes. Quando isto acontece, diz-se que eles estão ligados (*wired*). Como as interfaces estão especificando *commands* e *events*, elas são consideradas bidirecionais.

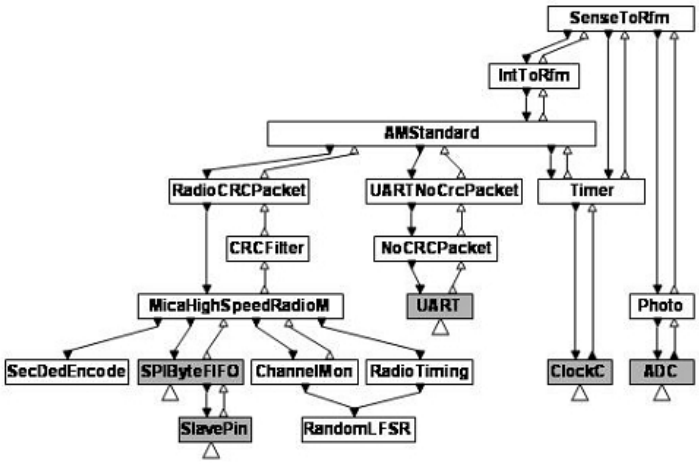

Figura 2.1: Grafo simplificado dos componentes nesC da aplicação SenseToRfm do TinyOS [19]

A Figura 2.1 apresenta um exemplo de grafo de componentes nesC de uma aplicação do TinyOS. A aplicação *SenseToRfm* coleta, periodicamente, sinais de luz e envia o valor em *broadcast* para outros sensores. Os nós do grafo são os componentes e as

arestas são interfaces. As setas para cima indicam o fluxo de *events* e as setas para baixo indicam o fluxo de *commands*.

2.3 Linguagem nesC

A linguagem nesC [11][14] é bastante similar a C e sua maior diferença é o modelo de ligação entre os componentes. O uso de componentes diminui o tempo de desenvolvimento de aplicações e permite a sua reusabilidade.

Existem dois tipos de componentes: Configuração e Módulo. As configurações definem como os componentes estão conectados e os módulos são as implementações das interfaces dos componentes.

2.3.1 Interfaces

As interfaces são pontos de acesso aos componentes e devem obedecer a um padrão e conter apenas as assinaturas dos *commands* e *events*. No exemplo a seguir é mostrada uma parte da interface para envio de pacotes do TinyOS.

```
(1)  interface Send {
(2)    command error_t send(message_t* msg, uint8_t len);
(3)    event void sendDone(message_t* msg, error_t error);
(4)    (...)
(5)  }
```

Quando se define o componente que usa ou provê esta interface, define-se qual lado do *split-phase* ele representa. Um provedor da interface `Send` precisa definir a função `send` (2) e sinalizar o evento `sendDone` (3). Já quem usar esta interface precisa definir um evento `sendDone` e realizar chamadas ao comando `send`.

2.3.2 Módulos

Os módulos contêm o código da aplicação implementando uma ou mais interfaces. Para exemplificar o uso dos módulos, o módulo `PeriodicSenseAndSendC` é apresentado a seguir. Este módulo `PeriodicSenseAndSendC` (1) lê informações do ambiente (19-21) e as envia em *broadcast* para outros sensores periodicamente (22-36). Ele provê a interface `StdControl` (2), ou seja, precisa implementar os comandos `start` (13) e `stop` (16) dessa interface. O componente usa as interfaces `Timer` (4), `Read` (5), `Send` (6) e `Packet` (7). Na definição do evento `readDone` (22) são feitas chamadas ao comando `send` (31) do componente de envio de pacotes. Quando este componente finalizar o envio, é lançado um evento `sendDone` indicando a conclusão da tarefa (processo

definido de *split-phase* na Seção 2.2). Assim, a aplicação pode liberar o acesso ao componente de envio de pacotes (38).

```
(1)  module PeriodicSenseAndSendC {
(2)    provides interface StdControl;
(3)    uses {
(4)      interface Timer<TMilli>;
(5)      interface Read<uint16_t>;
(6)      interface Send;
(7)      interface Packet;
(8)    }
(9)  }
(10) implementation {
(11)   message_t packet;
(12)   bool busy = FALSE;
(13)   command error_t StdControl.start() {
(14)     return call Timer.startPeriodic(1024);
(15)   }
(16)   command error_t StdControl.stop() {
(17)     return call Timer.stop();
(18)   }
(19)   event void Timer.fired() {
(20)     call Read.read();
(21)   }
(22)   event void Read.readDone(error_t err, uint16_t val) {
(23)     if (err != SUCCESS || busy) {
(24)       return;
(25)     }
(26)     else {
(27)       uint8_t payloadLen;
(28)       uint16_t* payload = (uint16_t*)call Packet.getPayload
     (\&packet, \&payloadLen);
(29)       if (payloadLen >= sizeof(uint16_t)) {
(30)         *payload = val;
(31)         if (call Send.send(\&packet, sizeof(uint16_t)) {
(32)           busy = TRUE;
(33)         }
(34)       }
(35)     }
(36)   }
(37)   event void Send.sendDone(message_t* msg, error_t error) {
(38)     busy = FALSE;
(39)   }
(40) }
```

As *tasks* descritas na Seção 2.2 podem estar presentes dentro da área de implementação do módulo. Um exemplo do uso de *task* pode ser visto a seguir. Uma *task* (6) para sinalizar a finalização do envio foi criada e passa a ser chamada durante a execução do comando `read` (10).

```
(1)  module PeriodicReaderC {
(2)    (...)
(3)  }
(4)  implementation {
(5)    (...)
(6)    task void readDoneTask() {
(7)      signal Read.readDone(SUCCESS, filterVal);
(8)    }
(9)    command error_t Read.read() {
(10)     post readDoneTask();
(11)     return SUCCESS;
(12)   }
(13) }
```

2.3.3 Configurações

Toda aplicação em nesC precisa de configuração cuja responsabilidade é a ligação entre os componentes. Na ligação, as interfaces providas por um componente são conectadas às interfaces usadas por outros.

Na definição da configuração também existe a opção de usar e prover interfaces. A configuração `BlinkAppC` (1) provê a interface `Init` (2).

```
(1)  configuration BlinkAppC {
(2)    provides interface Init;
(3)  }
(4)  implementation {
(5)    components MainC, BlinkC, LedsC;
(6)    components new TimerMilliC() as Timer0;
(7)    components new TimerMilliC() as Timer1;
(8)    components new TimerMilliC() as Timer2;
(9)
(10)   BlinkC -> MainC.Boot;
(11)   BlinkC.Timer0 -> Timer0;
(12)   BlinkC.Timer1 -> Timer1;
(13)   BlinkC.Timer2 -> Timer2;
(14)   BlinkC.Leds -> LedsC;
(15)   Init = BlinkC.Init;
(16) }
```

A implementação da configuração especifica os componentes que serão referenciados pelo atual componente e como eles serão conectados. Nesse exemplo, o

`BlinkAppC` referencia `MainC`, `BlinkC`, `LedsC` (5) e três instâncias de `TimerMilliC` (5-7).

2.4 TOSSIM

O TOSSIM [25] é um simulador baseado em eventos (e.g., interrupção de hardware, envio ou recebimento de mensagem). No momento da execução, estes eventos são colocados em uma fila e ordenados pelo tempo de ocorrência para serem executados posteriormente.

O simulador foi projetado especificamente para o TinyOS. Durante seu projeto os desenvolvedores se concentraram nas seguintes características [19]:

- **Escalabilidade**: suportar a simulação de uma rede com centenas de nós;
- **Completude**: capturar com exatidão todos os comportamentos observando o máximo de interações possíveis do sistema;
- **Fidelidade**: capturar as mais curtas interações entre os sensores é importante para avaliação e testes;
- **Validação de algoritmos**: validar as implementações dos algoritmos para os desenvolvedores verificarem se as aplicações irão executar com sucesso em um cenário real.

A escalabilidade é um ponto forte para uma simulação. Porém, com o TOSSIM, não é possível realizar testes de uma rede que contém sensores com aplicações diferentes. Todos os sensores da rede precisam executar a mesma aplicação, ou seja, uma rede com todos os nós idênticos.

Os desenvolvedores usam a biblioteca TOSSIM para escrever programas em Python ou C++ os quais configuram e executam as simulações. A biblioteca provê um conjunto de serviços de comunicação para outras aplicações. Isto permite que o TOSSIM se conecte, através de um *socket* TCP, a uma aplicação em Java, por exemplo, para fins de monitoração ou interação com a simulação.

O TOSSIM possui um sistema de depuração onde o usuário escolhe os meios de saída. Geralmente, o *console* é utilizado para exibição das mensagens de depuração, mas é possível escrever também a saída em um arquivo. Cada meio é associado a chaves (canais de depuração) que filtram as mensagens pertencentes apenas a elas. Explicitamente dentro da aplicação, o programador declara as mensagens de depuração e informa qual a chave que a possui. O exmplo a seguir mostra como é feita a definição de uma

mensagem de depuração (2). Desta forma, está sendo criada uma mensagem "Componente inicializado.\n" que será colocada na saída quando a chave "`Start`" for usada.

```
(1)  command error_t StdControl.start() {
(2)    dbg("Start", "Componente inicializado\n");
(3)    return call Timer.startPeriodic(1024);
(4)  }
```

Na linha (3) do código Python de um *script* de simulação, colocado a seguir, é adicionado o canal "Start" à saída padrão.

```
(1)  (...)
(2)  import sys
(3)  t.addChannel("Start", sys.stdout);
(4)  (...)
```

Sempre que o componente for inicializado, será colocado no *Console* (saída padrão) a mensagem "Componente inicializado".

2.5 Considerações Finais

Neste capítulo foram definidos os conceitos de sensores e redes de sensores sem fio. Os principais problemas das RSSFs foram apresentados: tolerância a falhas, escalabilidade, custos da produção, restrições de hardware, topologia, ambiente, meio de transmissão e consumo de energia.

O TinyOS foi apresentado como um sistema operacional para redes de sensores baseado em componentes que interagem entre si através de comandos e eventos. As aplicações do TinyOS são construídas em nesC a qual foram apresentados alguns exemplos simples de códigos.

As aplicações feitas para o TinyOS podem ser simuladas por uma biblioteca chamada TOSSIM. Este simulador tem a vantagem de suportar a escalabilidade da rede de sensores, porém não possui uma interface gráfica para visualização do comportamento da rede como um todo. Ele pode se comunicar com alguma outra aplicação, via *socket*, para realizar este trabalho.

3 Trabalhos Relacionados

Simulação é a forma mais comum de desenvolver e testar novos protocolos para uma rede de sensores. O objetivo é prever e modelar com exatidão o comportamento do ambiente do mundo real. Existem várias vantagens relacionadas ao uso de simulação: baixo custo, facilidade de implementação e praticidade de executar testes em redes de larga escala.

Neste capítulo, algumas ferramentas de simulação serão mostradas. Cada descrição apresenta as características principais, as vantagens e as desvantagens da ferramenta.

3.1 TinyViz

Como mencionado na Seção 2.4, o TOSSIM pode se conectar com outras aplicações para permitir o monitoramento e interação com a simulação. TinyViz [19][23] é a ferramenta de visualização do TOSSIM para versão 1.x do TinyOS. A sua interface gráfica foi implementada em Java e permite que o usuário analise e controle a execução da simulação.

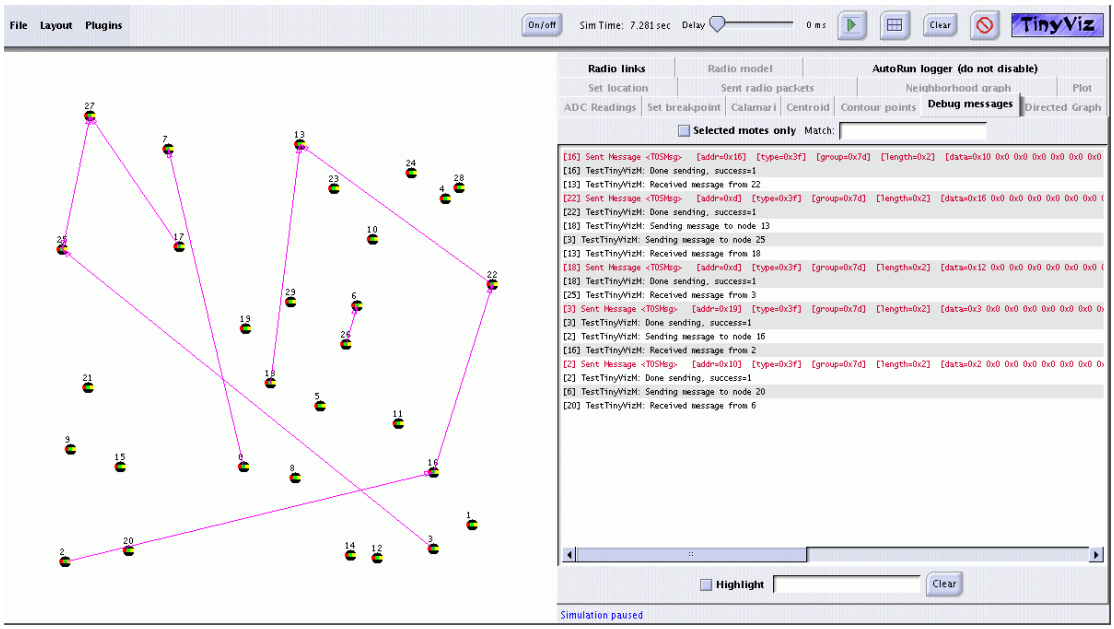

Figura 3.1: TinyViz conectado ao TOSSIM simulando uma RSSF [23]

A Figura 3.1 apresenta a interface gráfica do TinyViz exibindo a simulação de uma aplicação onde os sensores trocam mensagens entre si. Do lado esquerdo, os

sensores estão espalhados numa representação gráfica da rede. Do dado direito, as aplicações específicas (*plugins*) estão disponíveis para o usuário controlar o funcionamento do TinyViz.

O usuário tem a disposição na parte superior da aplicação recursos para iniciar, pausar e encerrar uma simulação, limpar o estado da representação gráfica da rede, e controlar a velocidade da visualização.

A arquitetura do TinyViz é flexível a ponto de permitir que desenvolvedores possam criar *plugins* de visualização e controle utilizando uma API provida pela *engine* TinyViz. Esta *engine* publica os eventos gerados pelo TOSSIM aos *plugins*. Isto permite que, por exemplo, um *plugin* visualize o tráfego das mensagens recebidas pelos sensores. Os *plugins* também podem controlar a simulação enviando comandos ao TOSSIM como, por exemplo, desabilitar um sensor da rede.

Dentre os *plugins* padrões do TinyViz, pode-se encontrar, além dos exemplos citados anteriormente, mensagens de depuração para análise do estado interno do sensor e *breakpoints* para pausar o TOSSIM.

3.2 NS-2

O NS-2 foi criado em 1989 [10][16] com o objetivo de simular o funcionamento das redes e se tornou bastante conhecido pelo fato de ser pioneiro e ter seu código aberto. Este simulador é baseado em eventos discretos e tem sua implementação bem modularizada permitindo facilmente sua extensibilidade. Dentre as adaptações feitas ao longo do tempo surgiu o suporte à simulação de redes de sensores.

O NS-2 é baseado em duas linguagens [5]: C++ e OTcl. O simulador foi escrito em C++ com o objetivo de utilizar o paradigma de orientação a objeto e de ter melhor desempenho. O usuário elabora *scripts* de comandos baseados numa extensão orientada a objetos de Tcl (*Tool Command Language*) chamada de OTcl.

A partir do *script* gerado pelo usuário, pode-se extrair informações da topologia da rede, protocolos usados, aplicações que se deseja simular e o formato de saída de dados do simulador. O OTcl pode utilizar os objetos criados no C++ através da ligação que cria um mapeamento de todos os objetos OTcl nos objetos de C++.

Por ser um simulador de eventos discretos, ele possui um agendador para gerenciar o momento que cada evento ocorrerá. Os eventos são objetos cujos atributos são: identificador único, horário agendado e um ponteiro para um objeto que irá realizar

o tratamento. Existe uma lista de eventos que são disparados um a um e, conseqüentemente, cada evento disparado invoca o objeto responsável por tratá-lo.

3.3 OMNet++

OMNet++ [17][28] é mais um sistema de simulação de eventos discretos com várias extensões para simulação de redes sem fio. Este simulador consiste de módulos simples escritos em C++ e de composições destes módulos. Os módulos simples são unidades indivisíveis na hierarquia de módulos capazes de se comunicar com outros componentes (módulos) através de suas portas de entrada e saída, definidas pela interface dos mesmos, quando estabelecidas conexões entre elas.

As composições de módulos podem ter apenas módulos simples ou uma combinação entre outras composições e módulos simples. Não há limite para a hierarquia de componentes. Dessa forma, há a possibilidade de se criar um nó da rede como sendo uma composição capaz de calcular o uso de energia e coletar dados do ambiente.

A criação da estrutura de composições é feita a partir da linguagem *Network Definition* (NED) [26] que é específica para definição de módulos e suas conexões a partir de uma interface gráfica ou textual.

O OMNet++ possui ainda uma interface gráfica. Dois tipos de interfaces estão disponíveis: TKENV e CMDENV. O CMDENV é uma interface de linha de comando ideal para simulações em *batch*.

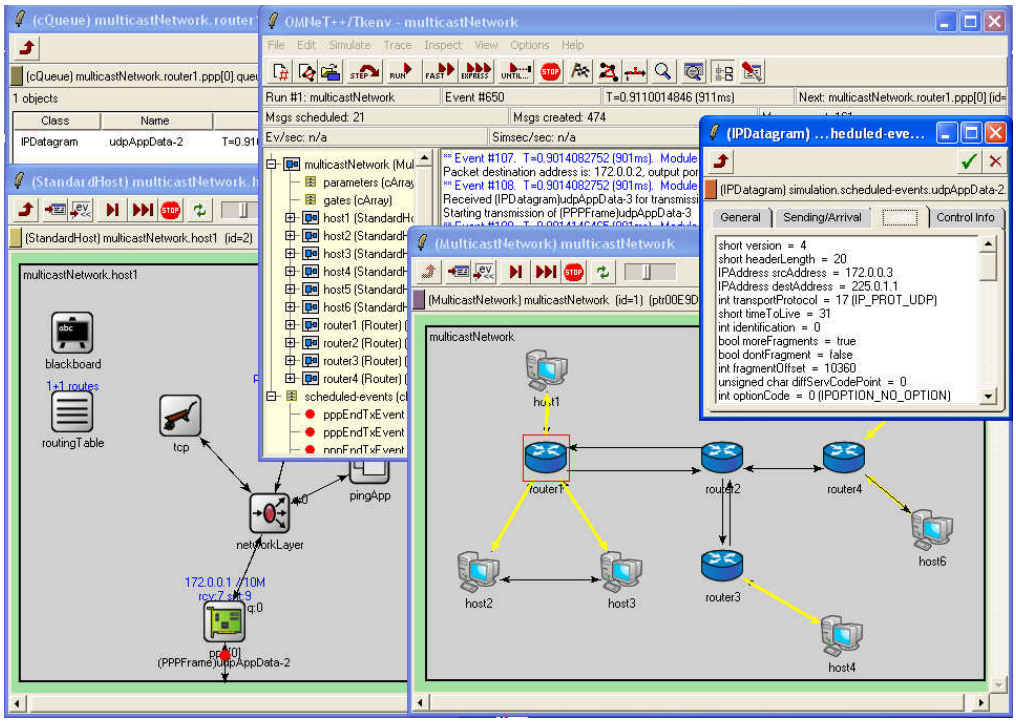

Figura 3.2: TKENV [17]

O TKENV (Figura 3.2) é a interface gráfica com o usuário. Nela, o usuário pode visualizar a animação do fluxo de mensagens trocadas na rede, abrir uma nova janela para análise das mensagens de *output* de um módulo individual ou em grupo e visualizar e editar o estado e o conteúdo de cada objeto da simulação.

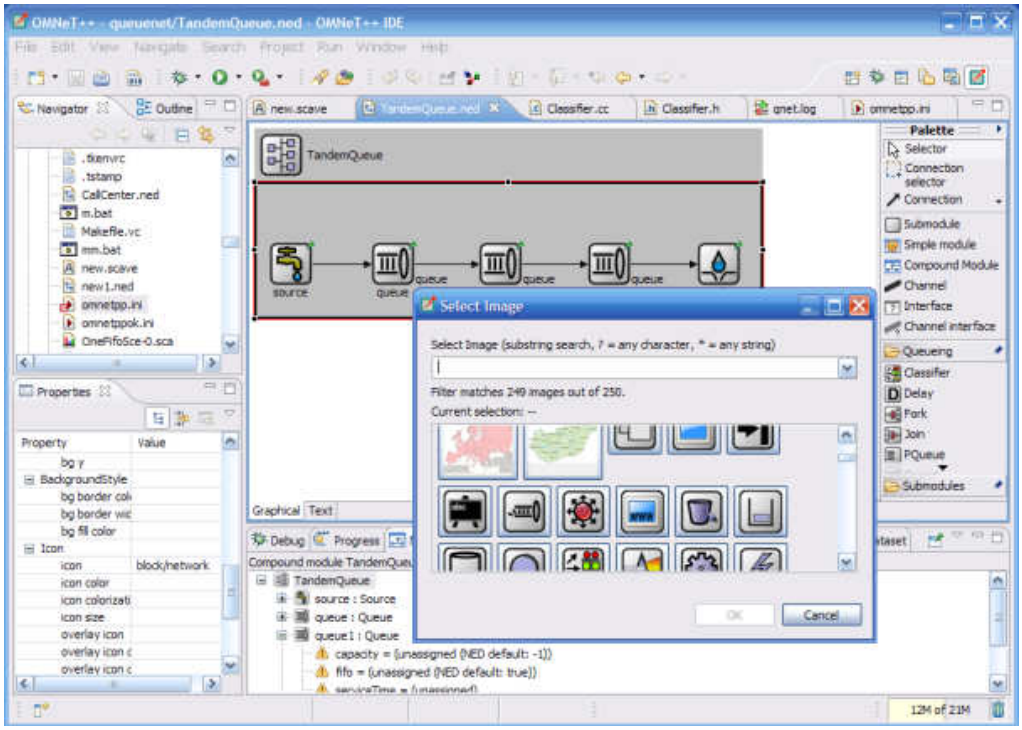

Figura 3.3: OMNet++ 4.0 IDE [17]

Para a última versão do OMNet++, foi desenvolvida um Ambiente Integrado de Desenvolvimento (IDE) baseado na plataforma Eclipse. Esta plataforma foi estendida com a criação, edição e visualização de modelos (arquivos NED e ini), execução em *batch* e análise dos resultados das simulações.

3.4 ATEMU

ATEMU [7][20] simula as operações de cada sensor da rede individualmente. Desta forma, ele permite que haja diferentes aplicações entre os sensores da rede durante a simulação. Ele usa uma estratégia de execução onde cada nó e cada dispositivo avançam em todas as rodadas de ciclo de *clock* para garantir que os nós, seus dispositivos e a comunicação estejam sempre sincronizados.

Inicialmente, o ATEMU, implementado em C, suporta apenas a simulação para sensores do tipo MICA2, mas sua arquitetura é flexível o suficiente para facilmente passar a suportar outras plataformas de hardware.

As informações de simulação extraídas do ATEMU possuem alto nível de confiança em relação à operação real da rede. Em conseqüência disto, o desempenho e a escalabilidade da rede foram comprometidos. Assim, há grande consumo de memória em simulações de redes com mais de 120 nós. Ele chega a ser 30 vezes mais lento que o TOSSIM [24].

É usado um arquivo XML de configuração que contém várias informações da rede e, especificamente, de cada sensor. Quando a topologia da rede é pequena este arquivo pode ser gerado por algum usuário, mas no caso de grandes topologias é mais conveniente escrever um *script* ou uma ferramenta para gerar o XML automaticamente. Uma configuração mínima precisa especificar a configuração do hardware, cópia do executável binário do software a ser executado e localização de cada nó.

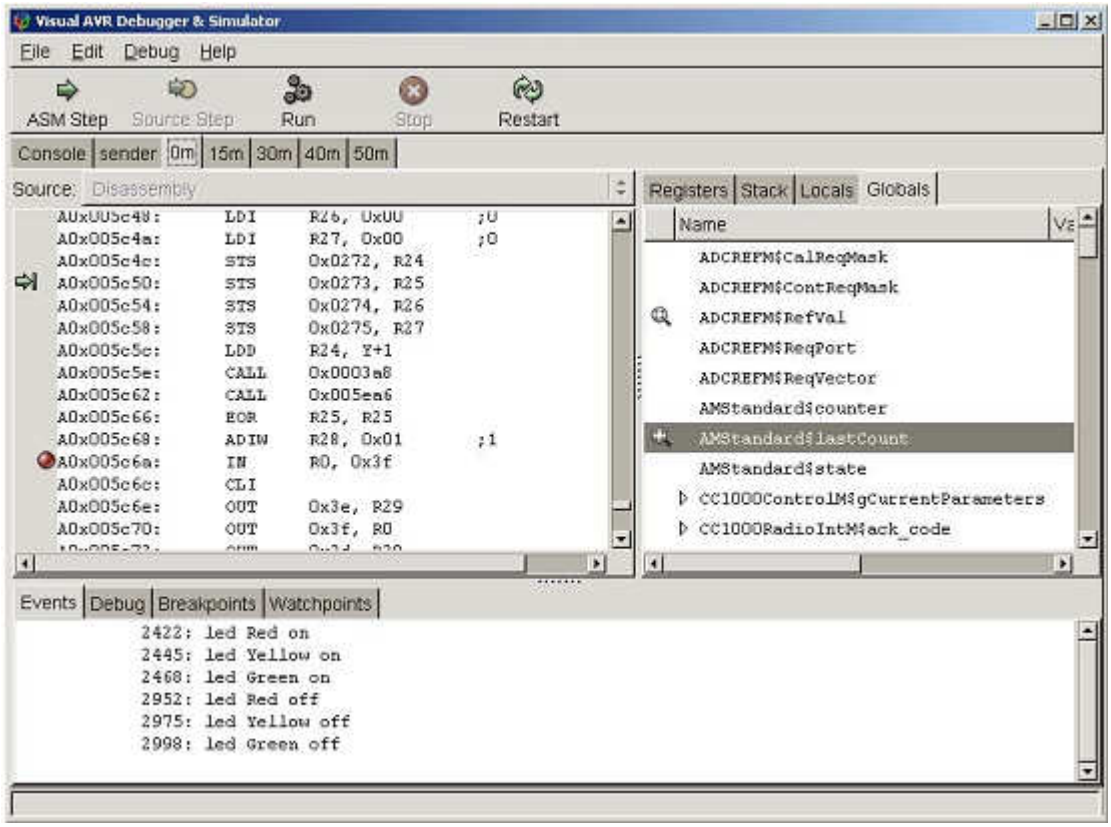

Figura 3.4: XATDB, a interface gráfica para o ATEMU [20]

Existe uma interface gráfica com o usuário chamada de XATDB (Figura 3.4) na qual é possível depurar e observar a execução do código *assembly* ou nesC através de *breakpoints*, *wathpoints* e execução passo a passo.

3.5 Avrora

O Avrora [8][24] é implementado em Java e, a exemplo de outros simuladores orientados a objeto, cria uma nova *thread* para cada nó da rede. Da mesma maneira que o ATEMU, o código é executado ciclo a ciclo, mas para melhorar o desempenho e escalabilidade os nós não são sincronizados em todos os ciclos, apenas quando necessário. Para tratar isso, o Avrora possui duas estratégias distintas para sincronização. A primeira é o intervalo de sincronização, onde é definida a freqüência que a sincronização ocorre. A segunda estratégia é a espera pelo vizinho que mantém uma estrutura de dados global para verificar o progresso de cada sensor.

Este simulador é uma aproximação do TOSSIM e do ATEMU. Ele une características presentes nos dois outros simuladores como simulação ciclo a ciclo e fila de eventos, além de eficiência na sincronização e *multi-thread*. Avrora é 50% mais lento que o TOSSIM e 20 vezes mais rápido que o ATEMU [24].

Quanto à escalabilidade da topologia, o Avrora pode suportar uma rede de até 1750 nós em um servidor com processador de dois núcleos de processamento. É possível realizar uma análise criteriosa da energia consumida dos sensores através da ferramenta chamada de AEON [2][13] que estende cada componente de hardware do Avrora monitorando o consumo durante a simulação. A análise de desempenho desta ferramenta mostra que o *overhead* adicionado ao Avrora é mínimo por monitorar apenas a mudança de estado dos componentes [13].

3.6 Considerações Finais

Este capítulo apresentou algumas ferramentas de simulação para RSSFs. Dentre os simuladores existentes, foram analisados o TinyViz, NS-2, OMNet++, ATEMU e Avrora.

O OMNet++, por exemplo, oferece uma boa interface gráfica com o usuário. No ATEMU, é possível depurar a execução das aplicações no nível das instruções em *assembly* ou em nesC. Uma característica considerada destaque no Avrora é a extensão da ferramenta para análise apurada do consumo de energia dos sensores.

Cada simulador mostrado possui pontos fortes e fracos em relação a escalabilidade, desempenho, precisão das informações extraídas da simulação e interface gráfica com o usuário.

4 TOSViz

O TOSViz é uma ferramenta de visualização do comportamento de RSSFs simuladas pelo TOSSIM. No TOSViz é possível visualizar e manipular uma topologia, visualizar as trocas de mensagens entre os sensores, obter informações da simulação (*e.g.*, dados estatísticos do uso de recursos), visualizar seqüência de ações e representar graficamente o comportamento da rede.

Detalhes do desenvolvimento da ferramenta são apresentados nas próximas seções seguindo as etapas tradicionais de desenvolvimento de software: definição dos requisitos, arquitetura, projeto e implementação.

4.1 Visão Geral

Com o objetivo de facilitar a visualização da simulação, o TOSViz interpreta as mensagens de depuração originadas do próprio TOSSIM que seriam apresentadas ao usuário em forma textual durante a simulação conforme mostrado na Figura 1.2.

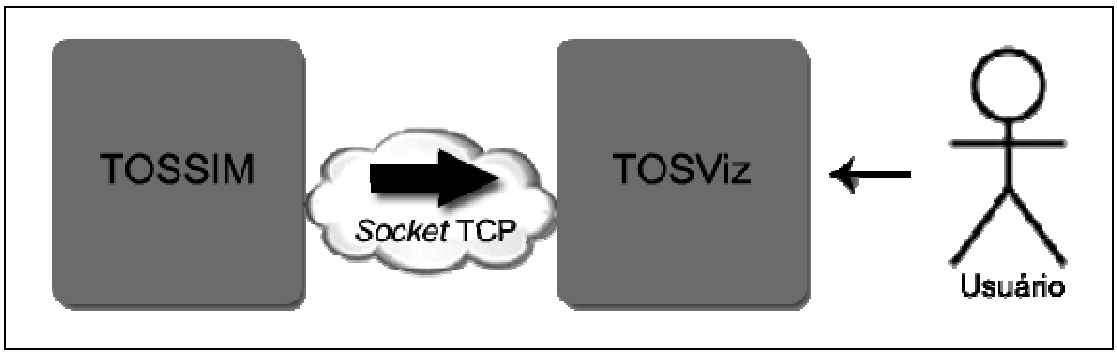

Figura 4.1: Comunicação entre o TOSSIM e o TOSViz

As mensagens são transferidas para a ferramenta de simulação através de um *socket* TCP conforme mostra a Figura 4.1. Isso explora a capacidade do TOSSIM de se conectar com outras aplicações citada na Seção 2.4.

4.2 Requisitos

Esta seção especifica os requisitos de alto nível que o TOSViz deve prover. Em particular, são identificados os requisitos funcionais, não-funcionais, casos de uso, restrições e premissas.

4.2.1 Prioridade dos Requisitos

Os requisitos do sistema recebem uma priorização estabelecida de acordo com os elementos descritos a seguir. Isto é importante para definição de prioridades na fase de desenvolvimento.

- **Essencial**: requisito necessário para o funcionamento do sistema;
- **Importante**: requisito que não interfere no funcionamento do sistema, mas sua ausência resulta num funcionamento não satisfatório; e
- **Desejável**: requisito que não interfere no funcionamento básico do sistema. Pode ser implementado por último ou não implementado.

4.2.2 Restrições e Premissas

Em projetos de software, geralmente existem limitações que interferem na forma que uma atividade é executada. Essas limitações são chamadas de restrições. Além disto, há a possibilidade de existir atividades que dependem de algumas premissas, ou seja, fatores considerados verdadeiros para tal atividade ser realizada com sucesso.

O desenvolvimento do **TOSViz** se baseia na existência das seguintes restrições/premissas:

[REST01] Formato das mensagens de depuração do TOSSIM

O TOSViz se baseia nas mensagens de depuração geradas pelo TOSSIM apresentadas na Tabela 4.1. Conforme apresentado na Seção 2.4, as chaves de depuração são canais que contêm um conjunto de mensagens de depuração. O usuário escolhe esses canais e o meio de saída das mensagens (e.g., *console*, arquivo).

Tabela 4.1: Mensagens de depuração do TOSSIM

Chave de Depuração	Formato da mensagem
AM	PACKET: Broadcasting packet to everyone.
AM	AM: Sending packet (id=<identificador do tipo da mensagem>, len=<tamanho da mensagem>) to <número de destino>
Gain	Getting link from <identificador de sensor> to <identificador de sensor> with gain <valor do ganho>
LedsC	LEDS: Led<identificador do LED> <*on* ou *off*>.
LI	newlen2 = <tamanho da mensagem>
LI	Sending seq: <número da seqüência>

LI	<bytes separados por espaço de 00 a FF>
LI	Got seq: <número da seqüência> from link: <identificador do sensor>
TossimPacketModelC	TossimPacketModelC: Init.init() called

[REST02] TOSSIM envia mensagens de depuração via *SerialForwarder*

É possível conectar o TOSSIM com alguma outra aplicação através do *SerialForwarder* para enviar dados.

4.2.3 Requisitos Funcionais

Os requisitos funcionais do TOSViz estão detalhados a seguir.

[RF01] Receber Mensagens do TOSSIM

A ferramenta deve ser capaz de estabelecer uma conexão com o TOSSIM por vez. Quando conectada, deve ser possível receber mensagens que definem as ações ocorridas na rede.

Prioridade: essencial

RFs relacionados: [RF02], [RF05], [RF12], [RF13]

[RF02] Iniciar Visualização

Assim que houver mensagens recebidas do TOSSIM, o TOSViz deve permitir que o usuário inicie a visualização da simulação. Ao iniciar, a ferramenta precisa interpretar as mensagens recebidas e executar as funcionalidades descritas em [RF05] e [RF08]. Além disso, é preciso guardar as informações necessárias para posterior execução das funcionalidades [RF12] e [RF13].

Prioridade: essencial

RFs relacionados: [RF01], [RF03], [RF04], [RF05], [RF08]

[RF03] Pausar Visualização

A qualquer momento que a simulação estiver sendo visualizada através da ferramenta, o usuário pode pausar a exibição das informações, ou seja, o sistema deixa de executar as funcionalidades [RF05] e [RF08].

Prioridade: importante

RFs relacionados: [RF02], [RF04]

[RF04] Parar Visualização

O sistema deve permitir que o usuário possa parar a simulação. Após parar uma simulação, o sistema já deve estar preparado para estabelecer mais uma nova simulação. Todos os dados da simulação parada são perdidos.

Prioridade: essencial

RFs relacionados: [RF02], [RF03]

[RF05] Exibir Mensagens de Simulação

O sistema deve exibir as mensagens resultantes das ações de simulação para o usuário. Os tipos das mensagens a serem exibidas são:

- **Inicialização**: inicialização do sensor;
- **Ativação e desativação** de LED: contém o identificador do LED e status de ligado ou desligado;
- **Exibição de topologia**: contém o identificador do sensor destino e o valor do ganho;
- **Envio em broadcast**: contém o tamanho do pacote, o identificador do tipo da mensagem e os dados;
- **Recebimento de mensagem**: mesmo que o anterior.

Todas as mensagens precisam estar associadas ao tempo de ocorrência na simulação e ao identificador do sensor que as originou.

Prioridade: essencial

RFs relacionados: [RF01], [RF03], [RF06], [RF07]

[RF06] Filtrar Mensagens de Simulação

O sistema deve permitir que as mensagens exibidas sejam filtradas pelo usuário. As opções de filtro são: tipo da mensagem e sensor

Prioridade: importante

RFs relacionados: [RF05]

[RF07] Limpar Mensagens de Simulação

O sistema deve permitir que as mensagens exibidas sejam excluídas. Desta forma, não é mais possível recuperá-las.

Prioridade: desejável

RFs relacionados: [RF05]

[RF08] Exibir Topologia de Visualização

O sistema deve exibir os sensores presentes na simulação. A posição de cada sensor deve respeitar a topologia importada através do [RF11]. Se nenhuma topologia tiver sido carregada na aplicação, a posição de cada sensor será gerada aleatoriamente pelo sistema. Com o passar do tempo de simulação, o sistema deve exibir envio de mensagens em *broadcast* e troca de mensagens entre os sensores.

Prioridade: importante

RFs relacionados: [RF01], [RF09], [RF10], [RF11]

[RF09] Alterar Topologia de Visualização

O sistema deve permitir que a posição de cada sensor possa ser alterada de acordo com a preferência do usuário.

Prioridade: importante

RFs relacionados: [RF08], [RF10], [RF11]

[RF10] Exportar Topologia de Visualização

O sistema deve permitir a exportação da topologia atual para arquivo. O arquivo deve conter apenas a posição de cada sensor presente na rede atual.

Prioridade: importante

RFs relacionados: [RF08], [RF09], [RF11]

[RF11] Importar Topologia de Visualização

O sistema deve permitir a importação de um arquivo de topologia. O arquivo deve conter apenas a posição de cada sensor presente na rede atual. Se durante a simulação for preciso criar mais sensores do que a quantidade presente na topologia importada, as posições dos novos sensores serão geradas aleatoriamente pelo sistema.

Prioridade: importante

RFs relacionados: [RF08], [RF09], [RF10]

[RF12] Exibir Dados da Topologia da Rede

O sistema deve ser capaz de exibir as possíveis conexões do sensor selecionado com outros nós da rede e os ganhos/níveis de interferência correspondentes.

Prioridade: essencial

RFs relacionados: [RF01], [RF13]

[RF13] Exibir Dados Estatísticos

O sistema deve ser capaz de exibir os dados de expectativa de consumo dos recursos relativos a cada sensor da rede. Os dados a serem exibidos são:

- **Mensagens**: quantidade de mensagens enviadas e recebidas e quantidade de bytes enviados e recebidos;
- **LEDs**: tempo total que cada LED do sensor ficou acesso;
- **Consumo estimado**: CPU, transmissão, recebimento, LEDs e total.

Prioridade: essencial

RFs relacionados: [RF01], [RF12]

4.2.4 Requisitos Não Funcionais

Os requisitos não-funcionais são detalhados a seguir.

[RNF01] Facilidade de uso

O sistema deve apresentar uma interface gráfica simples, com boa usabilidade e de fácil aprendizado.

Prioridade: importante

[RNF02] Portabilidade

O sistema deve funcionar em sistemas operacionais baseados em Linux.

Prioridade: essencial

4.2.5 Casos de uso

A Tabela 4.2 apresenta os atores identificados no sistema.

Tabela 4.2: Atores do TOSViz

Ator	Descrição
Temporizador	Periodicamente verifica se alguma mensagem recebida pode ser consumida/processada pelo TOSViz.
TOSSIM	Simulador de aplicações para RSSF.
Usuário	Usuário final do sistema que simula a aplicação pelo TOSSIM e visualiza a simulação pelo TOSViz.

O diagrama da Figura 4.2 define os casos de uso (UC) do TOSViz.

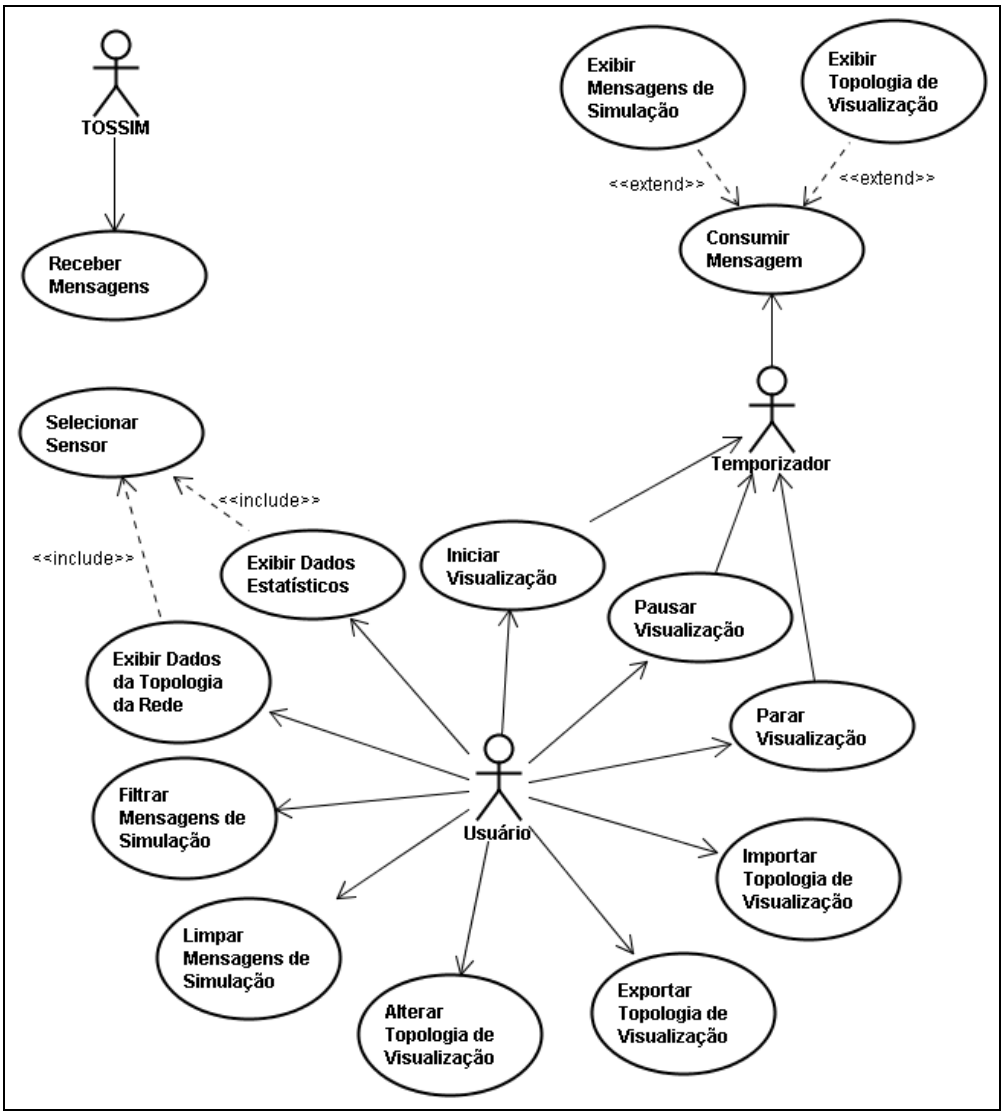

Figura 4.2: Diagrama de casos de uso do TOSViz

Os casos de uso identificados na Figura 4.2 são descritos a seguir:

[UC01] Receber Mensagens

Descrição: recebe as mensagens de depuração do TOSSIM

Ator: TOSSIM.

Pré-condições: não há.

Pós-condições: as mensagens recebidas precisam estar presentes em uma lista e prontas para serem consumidas/processadas.

Requisitos Atendidos: [RF01]

Casos de uso relacionados: não há.

Fluxo de eventos principal

1. Sistema aguarda a solicitação do TOSSIM para estabelecer uma conexão via *SerialForwarder*.

2. TOSSIM envia uma solicitação para se conectar ao sistema.
3. Sistema estabelece uma conexão com o TOSSIM.
4. Para cada mensagem enviada pelo TOSSIM:
   4.1. Sistema verifica o tipo de mensagem recebida. [FE01]
   4.2. Sistema guarda a mensagem numa lista para ser consumida posteriormente.
5. TOSSIM finaliza a conexão com o sistema.
6. Sistema permite que o usuário inicie a visualização da simulação. [FA01]
7. Fim do caso de uso.

Fluxos alternativos

[FA01] Sistema não recebe mensagens do TOSSIM

1. Se nenhuma mensagem for recebida durante a conexão com o TOSSIM, o sistema não permite que o usuário inicie a visualização da simulação.
2. Sistema aguarda uma nova conexão voltando para o passo 1 do fluxo principal.

Fluxos de erro

[FE01] Sistema recebe mensagem de tipo inválido

1. Sistema descarta a mensagem recebida que possui um tipo desconhecido.
2. Sistema recebe a próxima mensagem e executa o passo 4.1 do fluxo principal.

[UC02] Iniciar Visualização

Descrição: sistema começa a consumir as mensagens recebidas pelo TOSSIM e exibir as informações da simulação para o usuário.

Ator: usuário e temporizador.

Pré-condições: existir mensagem na lista aguardando ser consumida.

Pós-condições: as mensagens consumidas não devem mais estar presentes na fila.

Requisitos Atendidos: [RF02]

Casos de uso relacionados: não há.

Fluxo de eventos principal

1. Usuário pede para o sistema iniciar a visualização da simulação.
2. Sistema informa ao temporizador que as mensagens recebidas podem ser consumidas.
3. Sistema permite que o usuário possa pausar ou parar a simulação.

4. Fim do caso de uso.

[UC03] Pausar Visualização

Descrição: sistema congela a visualização da simulação.

Ator: usuário e temporizador.

Pré-condições: visualização da simulação ter sido iniciada.

Pós-condições: sistema congela a visualização da simulação.

Requisitos Atendidos: [RF03]

Casos de uso relacionados: não há.

Fluxo de eventos principal

1. Usuário pede para o sistema pausar a visualização da simulação.
2. Sistema informa ao temporizador que deixe de consumir as mensagens, porém guarde o tempo atual da simulação para continuá-la posteriormente.
3. Sistema permite que o usuário possa voltar a executar ou parar a simulação.
4. Fim do caso de uso.

[UC04] Parar Visualização

Descrição: sistema para a visualização da simulação e se prepara para uma nova simulação.

Ator: usuário e temporizador.

Pré-condições: visualização da simulação ter sido iniciada ou pausada.

Pós-condições: todas as informações da simulação precisam estar apagadas.

Requisitos Atendidos: [RF04]

Casos de uso relacionados: não há.

Fluxo de eventos principal

1. Usuário pede para o sistema parar a visualização da simulação.
2. Sistema informa ao temporizador que deixe de consumir as mensagens e reiniciar o tempo da simulação para iniciar uma nova visualização posteriormente.
3. Sistema apaga as mensagens da simulação, as estatísticas de uso dos recursos de cada sensor, a topologia de rede, a topologia da visualização dos sensores e as mensagens recebidas presentes na lista.
4. Fim do caso de uso.

[UC05] Exibir Mensagens de Simulação

Descrição: sistema exibe as mensagens resultantes das ações da simulação.

Ator: temporizador.

Pré-condições: alguma ação precisa ter acontecido como resultado da interpretação da mensagem consumida. As ações são: inicialização, mudança de estado do LED, envio e recebimento de mensagens e definição da topologia.

Pós-condições: as ações precisam estar detalhadas para o usuário.

Requisitos Atendidos: [RF05]

Casos de uso relacionados: [UC15]

Fluxo de eventos principal

1. Após a execução do [UC15], o temporizador pode exibir as mensagens de simulação para o usuário.
2. Sistema reúne informações para detalhar e exibir as ações da simulação para o usuário. O detalhamento da mensagem é composto pelo identificador do sensor, tempo de ocorrência e dados específicos de cada tipo de mensagem.
3. O identificador do sensor torna-se disponível nas opções de filtro disponíveis no [UC06].
4. Fim do caso de uso.

[UC06] Filtrar Mensagens de Simulação

Descrição: usuário define um filtro para exibição das mensagens de simulação.

Ator: usuário.

Pré-condições: não há.

Pós-condições: as mensagens de simulação precisam ser exibidas de acordo com o filtro definido pelo usuário.

Requisitos Atendidos: [RF06]

Casos de uso relacionados: não há.

Fluxo de eventos principal

1. Usuário informa que deseja realizar um filtro nas mensagens de simulação.
2. Sistema exibe as opções de filtro presentes no [RF06]. [FA01]
3. Usuário escolhe os tipos de mensagens que serão exibidas.
4. Usuário escolhe exibir mensagens de um ou todos ou sensores.
5. Usuário confirma operação.

6. Sistema exibe as mensagens de simulação respeitando o filtro indicado pelo usuário.
7. Fim do caso de uso.

Fluxos alternativos

[FA01] Usuário cancela a operação

1. Usuário cancela a operação.
2. Sistema permanece com as mesmas opções de filtro escolhidas anteriormente.
3. Fim do caso de uso.

[UC07] Limpar Mensagens de Simulação

Descrição: as mensagens de simulação geradas até o momento são apagadas permanentemente.

Ator: usuário.

Pré-condições: não há.

Pós-condições: não há mensagem de simulação a ser exibida para o usuário.

Requisitos Atendidos: [RF07]

Casos de uso relacionados: não há.

Fluxo de eventos principal

1. Usuário informa que deseja realizar limpar as mensagens de simulação.
2. Sistema limpa a lista de mensagens de simulação resultantes das ações da rede disponíveis para o usuário.
3. Fim do caso de uso.

[UC08] Exibir Topologia de Visualização

Descrição: sistema exibe a topologia de visualização junto com as ações de cada sensor (alteração do status dos LEDs, envio em *broadcast* e recebimento de mensagens).

Ator: temporizador.

Pré-condições: alguma ação precisa ter acontecido como resultado da interpretação da mensagem consumida. As ações são: mudança de estado do LED, envio e recebimento de mensagens.

Pós-condições: a topologia precisa ser atualizada com a ação exibida para o usuário.

Requisitos Atendidos: [RF08]

Casos de uso relacionados: [UC15]

Fluxo de eventos principal

1. Após a execução do [UC15], o temporizador pode exibir a topologia de visualização e ação do sensor para o usuário.
2. Caso o sensor que realizou a ação não exista na simulação, o sistema o exibe para o usuário. O identificador do sensor é disponibilizado para seleção que ocorrerá em [UC14] Sua posição na topologia de visualização é criada aleatoriamente pelo sistema. [FA01]
3. Sistema atualiza o estado do sensor de acordo com a ação realizada.
4. Sistema permite que o usuário exporte a topologia de visualização para arquivo.
5. Fim do caso de uso.

Fluxos alternativos

[FA01] Existe uma topologia importada anteriormente

1. Se o usuário estiver realizado uma importação de topologia e o identificador do sensor está presente junto com a respectiva posição, o sistema exibe o sensor respeitando sua posição importada.
2. Sistema executa o passo 3 do fluxo principal.

[UC09] Alterar Topologia de Visualização

Descrição: permite que o usuário altere as posições dos sensores presentes na visualização da simulação.

Ator: usuário.

Pré-condições: existir algum sensor ativo na rede.

Pós-condições: o sensor deve estar alocado na sua nova posição escolhida pelo usuário.

Requisitos Atendidos: [RF09]

Casos de uso relacionados: não há.

Fluxo de eventos principal

1. Usuário seleciona o sensor desejado.
2. Usuário escolhe nova localização para o sensor.
3. Sistema guarda a nova localização do sensor.
4. Sistema exibe o sensor na sua nova localização.
5. Fim do caso de uso.

[UC10] Exportar Topologia de Visualização

Descrição: as posições dos sensores presentes na visualização da simulação são armazenadas em um arquivo.

Ator: usuário.

Pré-condições: existir algum sensor ativo na rede.

Pós-condições: arquivo exportado com as informações da topologia de visualização.

Requisitos Atendidos: [RF10]

Casos de uso relacionados: não há.

Fluxo de eventos principal

1. Usuário informa que deseja exportar a atual topologia de visualização para arquivo.
2. Sistema solicita ao usuário o diretório e o nome que o arquivo será salvo. [FA01]
3. Usuário informa diretório e nome do arquivo.
4. Sistema salva no arquivo as posições dos sensores presentes na topologia associando-as ao identificador desses sensores. [FE01]
5. Fim do caso de uso.

Fluxos alternativos

[FA01] Usuário cancela a operação

1. Usuário cancela a operação.
2. Sistema permanece com as mesmas opções de filtro escolhidas anteriormente.
3. Fim do caso de uso.

Fluxos de erro

[FE01] Erro ao salvar o arquivo

1. Sistema não salva arquivo devido a problemas de entrada/saída.
2. Sistema informa o usuário do erro.
3. Fim do caso de uso.

[UC11] Importar Topologia de Visualização

Descrição: as posições dos sensores presentes na próxima visualização da simulação são importadas a partir de um arquivo.

Ator: usuário.

Pré-condições: a visualização não ter sido iniciada e nenhum outro arquivo tenha sido importado anteriormente para a próxima visualização da simulação.

Pós-condições: sensores posicionados de acordo com a topologia importada.

Requisitos Atendidos: [RF11]

Casos de uso relacionados: não há.

Fluxo de eventos principal

1. Usuário informa que deseja importar uma topologia de visualização para a próxima visualização de simulação.
2. Sistema solicita ao usuário o arquivo será importado. [FA01]
3. Usuário informa o arquivo.
4. Sistema carrega as posições dos sensores presentes no arquivo. [FE01]
5. Fim do caso de uso.

Fluxos alternativos

[FA01] Usuário cancela a operação

1. Usuário cancela a operação.
2. Sistema permanece com as mesmas opções de filtro escolhidas anteriormente.
3. Fim do caso de uso.

Fluxos de erro

[FE01] Arquivo inválido

1. Sistema identifica que o arquivo não possui um formato esperado.
2. Sistema informa o usuário do erro.
3. Fim do caso de uso.

[UC12] Exibir Dados da Topologia da Rede

Descrição: o sistema exibe informações da topologia da rede relativa apenas ao sensor selecionado.

Ator: usuário.

Pré-condições: a visualização da simulação ter sido iniciada e existir sensores ativos na rede.

Pós-condições: informações de topologia do sensor selecionado exibidas para o usuário.

Requisitos Atendidos: [RF12]

Casos de uso relacionados: [UC15]

Fluxo de eventos principal

1. Usuário informa que deseja visualizar a topologia da rede.
2. Sistema executa [UC15].

3. Sistema exibe quais sensores que o sensor selecionado pode se comunicar e os ganhos/níveis de interferência correspondentes.
4. Fim do caso de uso.

[UC13] Exibir Dados Estatísticos

Descrição: o sistema exibe informações estatísticas de uso de recursos relativas ao sensor selecionado.

Ator: usuário.

Pré-condições: a visualização da simulação ter sido iniciada e existir sensores ativos na rede.

Pós-condições: informações estatísticas do sensor selecionado exibidas para o usuário.

Requisitos Atendidos: [RF13]

Casos de uso relacionados: [UC15]

Fluxo de eventos principal

1. Usuário informa que deseja visualizar os dados estatísticos de uso de recursos.
2. Sistema executa [UC15].
3. Sistema exibe as informações descritas no [RF13].
4. Fim do caso de uso.

[UC14] Selecionar Sensor

Descrição: seleção de um dos sensores presentes na visualização da simulação.

Ator: usuário.

Pré-condições: existir algum sensor ativo na rede.

Pós-condições: o sensor deve estar selecionado.

Requisitos Atendidos: [RF12], [RF13]

Casos de uso relacionados: [UC12], [UC13]

Fluxo de eventos principal

1. Sistema disponibiliza os sensores da rede.
2. Usuário seleciona um sensor.
3. Sistema marca sensor como selecionado.
4. Fim do caso de uso.

[UC15] Consumir Mensagem

Descrição: sistema consome uma mensagem disponível na lista de mensagens recebidas.

Ator: temporizador.

Pré-condições: deve existir alguma mensagem disponível para o tempo atual de simulação determinada pelo temporizador. O sistema deve estar permitido de seguir com a visualização.

Pós-condições: a mensagem consumida deve ser removida da lista de mensagens recebidas.

Requisitos Atendidos: [RF03], [RF05], [RF08], [RF12], [RF13]

Casos de uso relacionados: [UC05], [UC08]

Fluxo de eventos principal

1. Temporizador solicita o consumo de mensagens informando o tempo atual de simulação.
2. Sistema consulta as mensagens.
3. Sistema verifica o tipo da mensagem e extrai informações para posterior exibição da topologia da rede ou estatísticas de uso dos recursos dos sensores.
4. Sistema elimina a mensagem consumida da lista de mensagens recebidas.
5. Este caso de uso é estendido por [UC05] e [UC08].

Fluxos de erro

[FE01] Não existem mensagens para o momento

1. Se nenhuma mensagem relacionada ao tempo informado for encontrada, o caso de uso é finalizado.

4.3 Arquitetura

A arquitetura da solução foi elaborada para suportar o recebimento de mensagens do TOSSIM visto que este possui diversas mensagens de depuração para informar ao usuário as ações acontecidas durante a simulação.

A Figura 4.3 mostra a alternativa encontrada para enviar as mensagens via *SerialForwarder* para a ferramenta de visualização sem precisar alterar o código fonte do TOSSIM. O *SerialForwarder* [23] é uma ferramenta do TinyOS que permite a comunicação de sensores com aplicações sobre o TCP/IP. Então, foi usada a biblioteca do *SerialForwarder* na simulação visando encaminhar as mensagens de depuração do TOSSIM para o TOSViz.

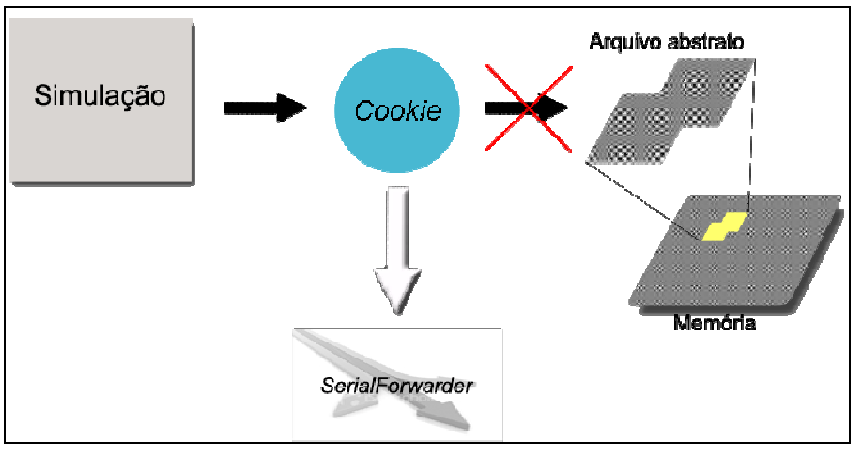

Figura 4.3: Redirecionamento das mensagens de depuração

Um meio de saída das mensagens de depuração pode ser escolhido na simulação. Na Seção 2.4 foi dito que o *Console* é o meio mais usado. No entanto, para o redirecionamento dos dados para o *SerialForwarder*, foi necessário escolher um arquivo "abstrato" como meio de saída. O arquivo foi chamado de "abstrato" por não existir fisicamente, evitando a escrita em disco e diminuição de desempenho. Ele é apenas um espaço alocado na memória que precisa ser acessado através de um *cookie*. Este último possui funções de acesso (escrita, leitura, busca e fechamento) as quais podem ser redefinidas. Desta forma, quando o TOSSIM tentar colocar uma mensagem no arquivo, a redefinição da função de escrita encaminha a mensagem para o *SerialForwarder*.

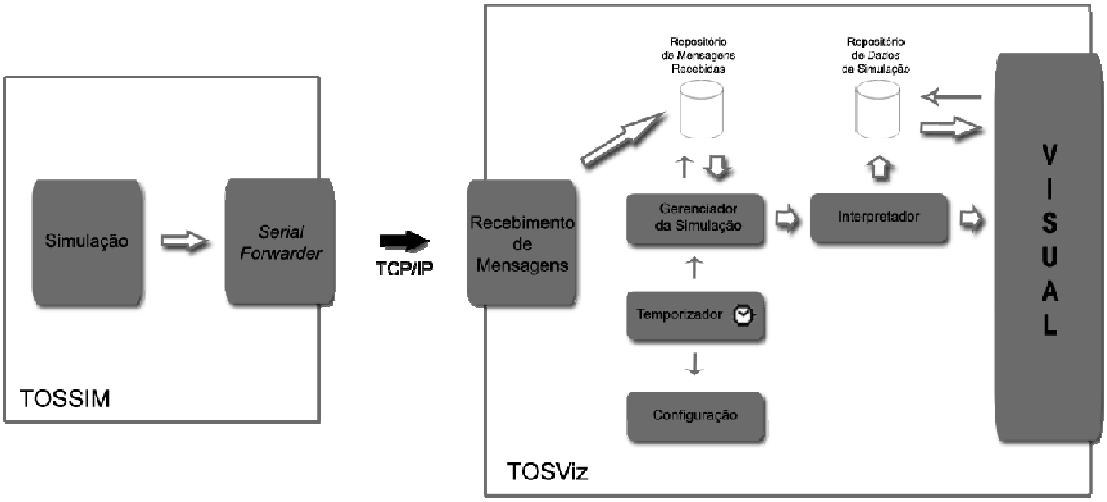

Figura 4.4: Arquitetura da solução

A Figura 4.4 mostra os relacionamentos entre os componentes e a interação entre TOSSIM e TOSViz. Os componentes que compõem o TOSViz são apresentados e descritos na Tabela 4.3.

Tabela 4.3: Componentes do TOSViz

Componente	Descrição/Responsabilidade
Configuração	Guarda parâmetros relativos à simulação (e.g., unidade de tempo).
Gerenciador da simulação	Verifica se as solicitações de consumo de mensagens recebidas podem ser atendidas respeitando o *status* da visualização (e.g., iniciada, pausada, parada).
Interpretador	Realiza o *parser* nas mensagens consumidas extraindo as informações necessárias para a visualização da simulação.
Recebimento de mensagens	Estabelece conexão com o TOSSIM, recebe as mensagens de depuração e as guarda no repositório de mensagens recebidas.
Repositório de dados da simulação	Armazena dados da simulação importantes (e.g., dados estatísticos do uso de recursos, topologia da rede) para posterior exibição para o usuário.
Repositório de mensagens recebidas	Armazena as mensagens recebidas do TOSSIM que serão consumidas pela visualização.
Temporizador	Periodicamente solicita o consumo das mensagens recebidas.
Visual	Exibe o passo a passo da simulação gráfica e textualmente. Recebe as requisições do usuário, consulta o repositório de dados da simulação e exibe as informações solicitadas.

A Tabela 4.4 apresenta as bibliotecas e *frameworks* utilizados no desenvolvimento da solução proposta.

Tabela 4.4: Bibliotecas e frameworks

Nome	Versão	Arquivos
Draw 2D	3.2.100	org.eclipse.draw2d_3.2.100.v20070529.jar
Quartz	1.6.0	commons-collections-3.1.jar commons-logging.jar commons-logging-api.jar jta.jar quartz-all-1.6.0.jar
SWT	3.3.2	org.eclipse.swt.gtk.linux.x86_3.4.0.v3448f.jar

		org.eclipse.swt_3.3.2.v3349d.jar
TinyOS	2.1	tinyos.jar

4.4 Projeto

Após a análise dos requisitos e elaboração da arquitetura, o sistema foi modelado seguindo o diagrama de classes da Figura 4.5 cujos elementos mostrados são descritos na Tabela 4.5.

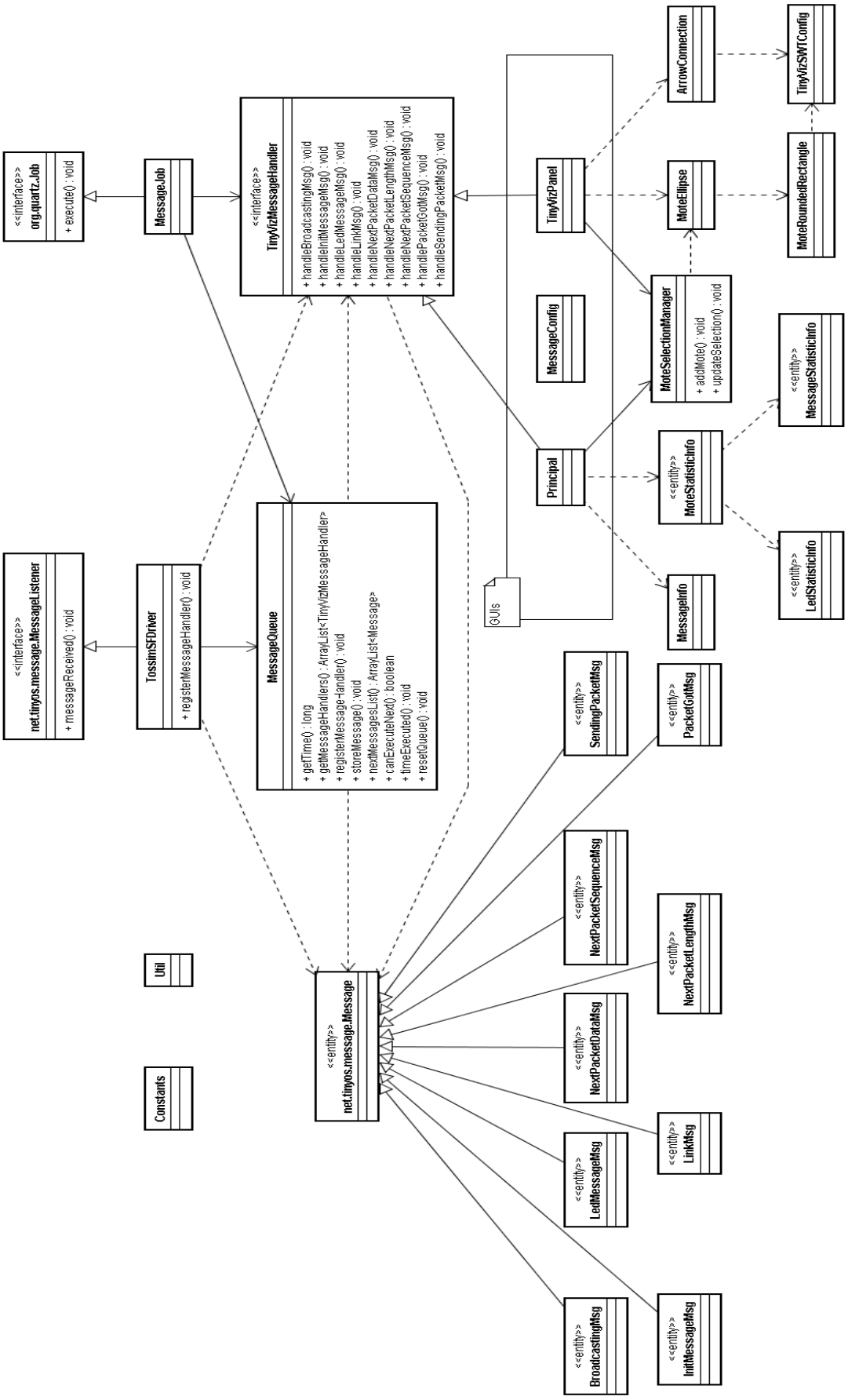

Figura 4.5: Diagrama de classes

Tabela 4.5: Descrição das classes e interfaces usadas na ferramenta

Classe/Interface	Descrição
net.tinyos.message.Message	Classe do TinyOS que representa a mensagem genérica trocada entre sensores ou entre sensor e computador.
BroadcastingMsg	Especialização da classe net.tinyos.message.Message em mensagem que significa envio em *broadcasting*.
InitMessageMsg	Especialização da classe net.tinyos.message.Message em mensagem que significa inicialização do componente de troca de pacotes (dados).
LedMessageMsg	Especialização da classe net.tinyos.message.Message em mensagem que significa alteração de estado do LED
LinkMsg	Especialização da classe net.tinyos.message.Message em mensagem que significa possível conexão entre dois sensores e o ganho/interferência existente.
NextPacketDataMsg	Especialização da classe net.tinyos.message.Message em mensagem que significa próximo byte daqueles que estão sendo enviados em uma mensagem.
NextPacketLengthMsg	Especialização da classe net.tinyos.message.Message em mensagem que significa tamanho do pacote que está sendo enviado.
NextPacketSequenceMsg	Especialização da classe net.tinyos.message.Message em mensagem que significa o número de seqüência para identificação do pacote enviado.
PacketGotMsg	Especialização da classe net.tinyos.message.Message em mensagem que significa recebimento do pacote.

SendingPacketMsg	Especialização da classe net.tinyos.message.Message em mensagem que significa o início do envio de um pacote.
org.quartz.Job	Interface pertencente a biblioteca Quartz. As atividades agendadas precisam implementar esta interface.
MessageJob	Agendamento para consumo de mensagens. Implementa a interface org.quartz.Job.
MessageQueue	Representa a lista de mensagens recebidas do TOSSIM. É responsável, também pelo controle do tempo da simulação.
TinyVizMessageHandler	Interface para consumo das mensagens. Todas as classes que sofrem alteração quando uma mensagem é consumida precisa implementar esta interface.
net.tinyos.message.MessageListener	Interface do TinyOS que permite o recebimento das mensagens enviadas pelo *SerialForwarder*.
TossimSFDriver	Classe que implementa a interface net.tinyos.message.MessageListener para recebimento das mensagens do *SerialForwarder*.
LedStatisticInfo	Contém dados estatísticos de LEDs relacionados a um sensor.
MessageStatisticInfo	Contém dados estatísticos de mensagens (e.g., envio, recebimento, CPU) relacionados a um sensor.
MoteStatisticInfo	Reúne os dados estatísticos de um sensor.
Constants	Contém as constantes utilizadas para implementação da ferramenta.
Util	Métodos genéricos do sistema.
ArrowConnection	Componente da visualização gráfica. Seta que significa um mensagem trocada entre dois sensores.
MessageConfig	Tela para escolha do filtro das mensagens.
MessageInfo	Reúne informações do último pacote trocado

	pelos sensores (e.g., tamanho, seqüência, dados, tipo).
MoteEllipse	Componente da visualização gráfica. Possui um MoteRoundedRectangle e muda a cor da borda em eventos relacionados ao sensor (e.g., *broadcasting*, seleção).
MoteRoundedRectangle	Componente da visualização gráfica. Representa o sensor com três LEDs.
MoteSelectionManager	Classe responsável pelo gerenciamento da seleção de um sensor e ações da interface relacionadas a seleção.
Principal	Tela principal da ferramenta que reúne as três áreas da simulação (e.g., gráfica, mensagens e informações do sensor)
TinyVizPanel	Painel de visualização gráfica.
TinyVizSWTConfig	Possui parâmetro para os componentes da visualização gráfica.

A Tabela 4.6 mostra a organização das classes e interfaces. Foram originados 5 (cinco) pacotes separados pelos conceitos de interface gráfica com o usuário, uso geral, dados estatísticos, mensagens e coleta, armazenamento e controle das mensagens recebidas do TOSSIM.

Tabela 4.6: Organização das classes e interfaces do sistema em pacotes

Pacote	Classes/Interfaces
br.cin.ufpe.tinyos.message	BroadcastingMsg InitMessageMsg LedMessageMsg LinkMsg NextPacketDataMsg NextPacketLengthMsg NextPacketSequenceMsg PacketGotMsg SendingPacketMsg
br.cin.ufpe.tinyos.sf	MessageJob

	MessageQueue TinyVizMessageHandler TossimSFDriver
br.cin.ufpe.tinyos.statistics	LedStatisticInfo MessageStatisticInfo MoteStatisticInfo
br.cin.ufpe.tinyos.util	Constants Util
br.cin.ufpe.tinyos.viz	ArrowConnection MessageConfig MessageInfo MoteEllipse MoteRoundedRectangle MoteSelectionManager Principal TinyVizPanel TinyVizSWTConfig

4.5 Implementação

A implementação da interface gráfica da ferramenta concentra-se em duas janelas: principal (ver Figura 4.6) e filtro de mensagens (ver Figura 4.11).

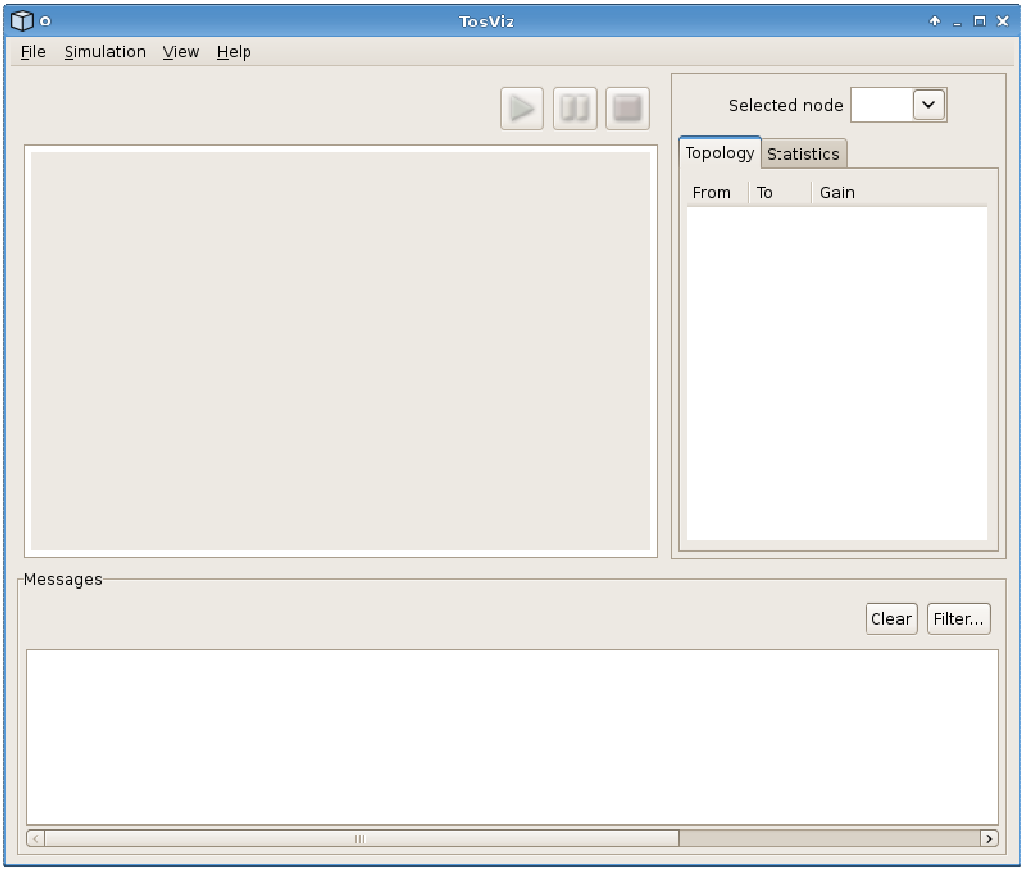

Figura 4.6: Janela principal

A janela principal está dividida em área de visualização gráfica, área de informações extras da simulação e área de mensagens da simulação. A área de visualização gráfica exibe os sensores espalhados no painel e suas ações associadas (e.g, ligar/desligar LED, enviar mensagens em *broadcast*, receber mensagens de outros sensores). Nela, o usuário pode controlar a continuação da simulação com os botões para iniciar, pausar e parar. O exemplo mostrado na Figura 4.7 possui uma rede composta de quatro sensores onde dois deles estão enviando mensagens.

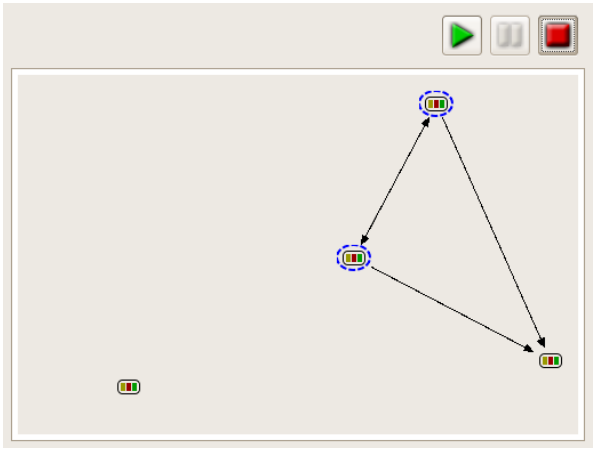

Figura 4.7: Área de visualização gráfica

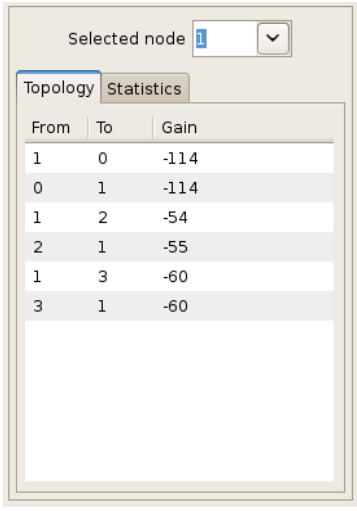

Figura 4.8: Área de informações da simulação (topologia)

A área de informações da simulação possui um campo de escolha do nó cujas informações serão exibidas nas duas abas disponíveis. A primeira aba (Figura 4.8) apresenta as possíveis conexões do sensor selecionado e a segunda aba (Figura 4.9) dispões das estatísticas de uso dos recursos.

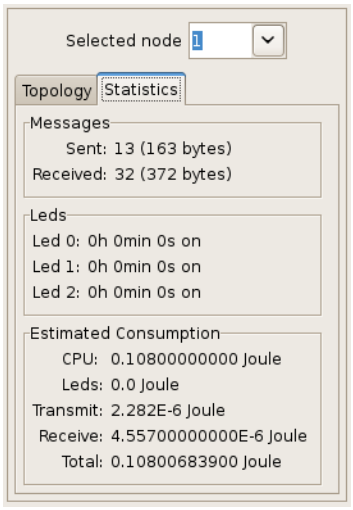

Figura 4.9: Área de informações da simulação (estatísticas)

A área de mensagens da simulação diferencia os tipos de mensagens por cores. Alguns exemplos dessas mensagens podem ser vistos na Figura 4.10. Nesta área, o usuário visualiza todo o *trace* da simulação que pode ser limpo ou filtrado.

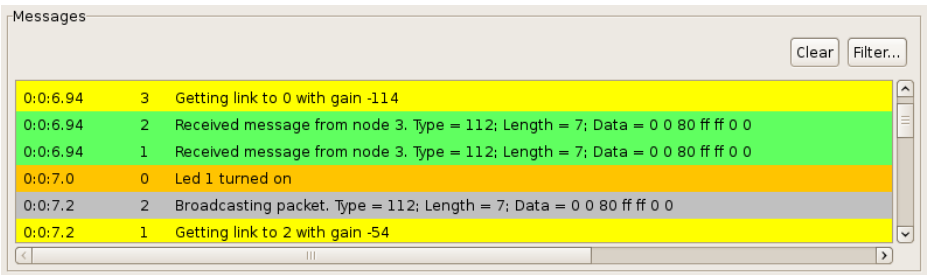

Figura 4.10: Área de mensagens da simulação

Figura 4.11: Janela de filtro das mensagens de simulação

A Figura 4.11 mostra a janela para escolha do filtro das mensagens. O filtro pode ser feito pelos tipos de mensagens e algum sensor específico. Se nenhum sensor for escolhido, mensagens de todos os sensores serão mostradas.

4.6 Considerações Finais

Este capítulo apresentou detalhes do TOSViz. Uma visão geral foi dada sobre o que é a ferramenta e como funciona a integração entre o TOSViz e o TOSSIM explicada com mais detalhes durante a definição da arquitetura na Seção 4.3.

Foram encontrados treze requisitos funcionais os quais deram origem a quinze casos de uso. Os atores identificados nos casos de uso foram o usuário, o TOSSIM e o temporizador.

Os casos de uso foram considerados nas fases de Análise e Projeto. A partir disso, as classes foram criadas e agrupadas seguindo conceitos de interface gráfica com o usuário, uso geral, dados estatísticos, mensagens e coleta, armazenamento e controle das mensagens recebidas do TOSSIM.

5 Avaliação da Nova Ferramenta

Uma topologia de rede com 10 sensores foi criada para avaliar o funcionamento do TOSViz em duas situações distintas. A primeira é o caso onde todos os sensores apenas trocam o estado de seus LEDs tornando-os acesos ou apagados (Blink). A outra situação, considerada mais próxima da realidade de RSSF, simula a existência de mensagens sendo trocadas constantemente (MViz). Nos dois exemplos, foram observados a utilização da memória e do processador pelo TOSViz

A avaliação foi feita utilizando uma máquina virtual executando o sistema operacional XubunTOS e com alocação de 512MB de memória. O computador utilizado pela máquina virtual possui sistema operacional Windows XP SP2, 1GB de memória RAM e processador Athlon 64 3200+.

A Tabela 5.1 possui o resultado dos testes com relação ao uso de memória (*Resident Set Size* - RSS) da ferramenta para as aplicações testadas usando a topologia com 10 nós. Além dos dois tipos de aplicações testadas, foi observado o uso da memória quando a ferramenta está ociosa. Neste caso, a maior quantidade de memória usada foi na situação onde o TOSViz aguarda conexão com o TOSSIM, pois constantemente a tentativa de conexão falha e em seguida uma nova tentativa é feita.

Tabela 5.1: Uso de memória do TOSViz para uma rede de 10 sensores

Aplicação	Utilização da memória	Utilização do processador
Nenhuma (aguardando conexão com o TOSSIM)	8150 KB	< 10%
Nenhuma (conectado com o TOSSIM)	2500 KB	< 10%
Blink	7700 KB	10% > uso > 50%
Mviz	5800 KB	> 50%

A saída gráfica da visualização para a aplicação dos LEDs, a Blink, não apresentou lentidão ou travamento do TOSViz por causa da baixa quantidade de mensagens recebidas do TOSSIM para indicar a alteração de *status* dos LEDs. Entretanto, ao tentar visualizar o desempenho da rede de topologia com 10 sensores executando a aplicação MViz , cujas trocas de mensagens são constantes, o uso do

processador foi maior que 50% em conseqüência da grande quantidade de mensagens recebidas do TOSSIM e consumidas pelo TOSViz. Para cada mensagem trocada entre sensores, a ferramenta precisa receber mais mensagens do simulador.

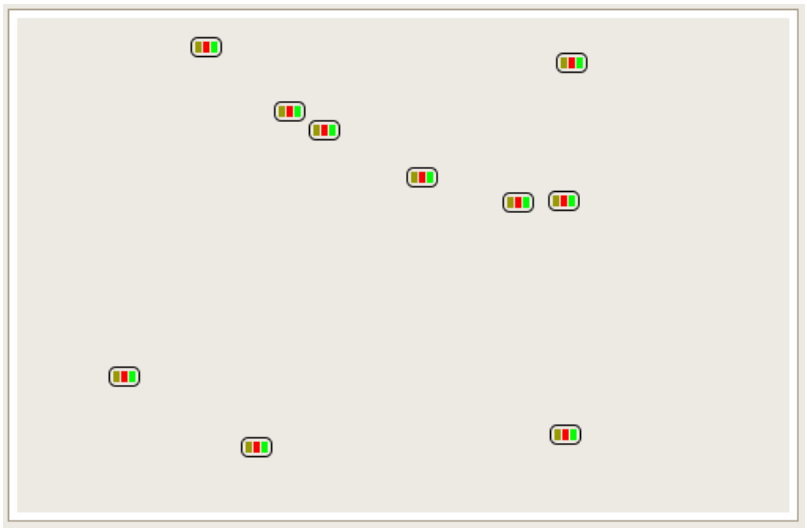

Figura 5.1: Teste do visualizador com uma aplicação que apenas altera o estado dos LEDs

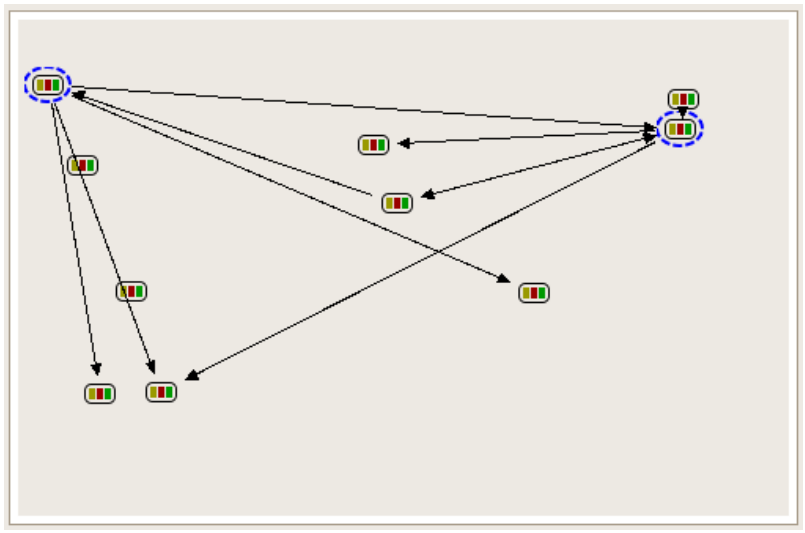

Figura 5.2: Teste do visualizador com uma aplicação que freqüentemente existem trocas de mensagens

5.1 Considerações Finais

Neste capítulo, foram realizados testes para análise da ferramenta proposta. Os resultados mostram que a ferramenta avaliada ainda precisa melhorar o desempenho quando ocorre recebimento excessivo de mensagens do TOSSIM.

6 Conclusão

Este trabalho de graduação apresentou o TOSViz, ferramenta criada para visualizar e manipular topologia de RSSF, visualizar trocas de mensagens entre os sensores, obter informações da simulação (*e.g.*, dados estatísticos do uso de recursos) e visualizar seqüência de ações. Baseado nessas funcionalidades iniciais, foi feito um levantamento de requisitos, arquitetura, análise e projeto e implementação do sistema.

O TOSViz permite aos desenvolvedores usarem o TOSSIM de forma mais agradável, mesmo sem substituir totalmente a análise textual feita sobre a saída da depuração da simulação. Também é possível realizar um filtro na visualização dos eventos gerados pela rede e, visando uma apresentação de forma clara, os eventos são destacados por cores de acordo com seus tipos. Esses fatores facilitam a análise do *trace* da simulação.

O usuário pode usar os recursos de importação/exportação de topologia da visualização. Assim, há a opção de ter uma topologia preferencial sempre que uma nova simulação for visualizada.

Outra característica interessante é o resumo de informações que podem ser adquiridas de um sensor da rede após o início da visualização da simulação. Dentre essas informações estão presentes a topologia e os dados estatísticos de utilização dos recursos, tais como: envio e recebimento de pacotes, tempo de utilização dos LEDs, estimativa do consumo de energia.

Um teste foi feito com TOSViz para mostrar o comportamento de uma rede cuja topologia possui 10 sensores usando uma aplicação onde ocorram freqüentemente troca de pacotes. A ferramenta proposta neste trabalho teve o seu objetivo cumprido quanto às funcionalidades levantadas e simplicidade de uso. Contudo, ainda existem limitações a serem superadas tais como o custo do consumo das mensagens recebidas do TOSSIM e escalabilidade da rede.

6.1 Trabalhos Futuros

Como trabalhos futuros, as seguintes atividades devem ser consideradas. Primeiro, realizar um estudo para tentar usar o caminho inverso da comunicação feita hoje entre TOSSIM e TOSViz. Isto permitiria ao usuário influenciar a simulação vista.

Segundo, otimização de desempenho e estimativas de consumo. Finalmente, a aplicação pode se transformar em um *plug-in* para a plataforma de desenvolvimento Eclipse.

7 Referências

[1] A Survey on Wireless Sensor Network Simulators. Disponível em: <http://tmtam.wordpress.com/2007/07/28/a-survey-on-wireless-sensor-network-simulators/>. Acesso em: 25/08/2008

[2] AEON. Disponível em: <http://ds.informatik.rwth-aachen.de/research/projects/aeon/>. Acesso em: 20/11/2008.

[3] Akyildiz, I.; Su, W.; Sankarasubramaniam, Y. & Cayirci, E. (2002), 'A survey on sensor networks', *IEEE Communications Magazine* **40**(8), 102--114.

[4] AlgoSenSim. Disponível em: <http://tcs.unige.ch/doku.php/code/algosensim/overview>. Acesso em: 26/08/2008.

[5] Altman, E. & Jiménez, T. (2003-2004), 'NS Simulator for beginners'.

[6] Arampatzis, T.; Lygeros, J. & Manesis, S. (2005), A Survey of Applications of Wireless Sensors and Wireless Sensor Networks, in J. Lygeros, ed.,'Proc. IEEE International Symposium on Mediterrean Conference on Control and Automation Intelligent Control', pp. 719--724.

[7] ATEMU. Disponível em: <http://www.hynet.umd.edu/research/atemu/>. Acesso em: 20/11/2008.

[8] Avrora. Disponível em: <http://compilers.cs.ucla.edu/avrora/>. Acesso em: 20/11/2008.

[9] Castalia. Disponível em: <http://castalia.npc.nicta.com.au/>. Acesso em: 26/08/2008.

[10] Curren, D. (2007), 'A Survey of Simulation in Sensor Networks', Technical report, University of Binghamton.

[11] Gay, D.; Levis, P.; von Behren, R.; Welsh, M.; Brewer, E. & Culler, D. (2003), The nesC language: A holistic approach to networked embedded systems, *in* 'PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation', ACM, New York, NY, USA, pp. 1--11.

[12] GTNetS. Disponível em: <http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/index.html>. Acesso em: 26/08/2008.

[13] Landsiedel, O.; Wehrle, K. & Gotz, S. (2005), Accurate Prediction of Power Consumption in Sensor Networks, in 'Proc. EmNetS-II Embedded Networked Sensors The Second IEEE Workshop on', pp. 37--44.

[14] Levis, P. (2006), 'TinyOS Programming'.

[15] Mannasim. Disponível em: <http://www.mannasim.dcc.ufmg.br>. Acesso em: 26/08/2008.

[16] NS-2. Disponível em: <http://www.isi.edu/nsnam/ns/>. Acesso em: 26/08/2008.

[17] OMNet++. Disponível em: <http://www.omnetpp.org/>. Acesso em: 26/08/2008.

[18] Pantazis, N.; Pantazis, N. & Vergados, D. (2007), 'A survey on power control issues in wireless sensor networks', *IEEE Communications Surveys & Tutorials* **9**(4), 86--107.

[19] Philip Levis, Nelson Lee, M. W. D. C. (2003), 'TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications', Technical report.

[20] Polley, J.; Blazakis, D.; McGee, J.; Rusk, D. & Baras, J. S. (2004), ATEMU: a fine-grained sensor network simulator, *in* 'Proc. First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks IEEE SECON 2004', pp. 145--152.

[21] Qualnet. Disponível em: <http://www.scalable-networks.com/>. Acesso em: 26/08/2008.

[22] SENSE. Disponível em: <http://www.ita.cs.rpi.edu/sense/index.html>. Acesso em: 26/08/2008.

[23] TinyOS. Disponível em: <http://www.tinyos.net/>. Acesso em: 18/08/2008.

[24] Titzer, B. L.; Lee, D. K. & Palsberg, J. (2005), Avrora: scalable sensor network simulation with precise timing, *in* 'Proc. Fourth International Symposium on Information Processing in Sensor Networks IPSN 2005', pp. 477--482.

[25] TOSSIM. Disponível em: <http://docs.tinyos.net/index.php/TOSSIM>. Acesso em: 18/08/2008.

[26] Varga, A. (1998), Parametrized Topologies for Simulation Programs, *in* 'Proc. Western Multiconference on Simulation (WMC'98), Communication Networks and Distributed Systems (CNDS'98)'.

[27] VisualSense. Disponível em: <http://ptolemy.berkeley.edu/visualsense/>. Acesso em: 26/08/2008.

[28] Xian, X.; Shi, W. & Huang, H. (2008), Comparison of OMNET++ and other simulator for WSN simulation, *in* 'Proc. 3rd IEEE Conference on Industrial Electronics and Applications ICIEA 2008', pp. 1439--1443.

O guia do usuário está dividido em três partes para que o usuário não sinta dificuldade na utilização do TOSViz. Primeiro, os elementos do painel de simulação e as mensagens de simulação são detalhados. Em Seguida, é apresentado um passo a passo para iniciar a visualização de um exemplo. Por último, o uso das demais funcionalidades é explicado.

Elementos do TOSViz

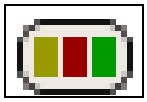

Figura A.1: Sensor. Internamente, ele possui três retângulos que equivalem aos LEDs. No decorrer da simulação, é exibido o *status* de ligado ou desligado.

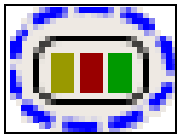

Figura A.2: Sensor enviando mensagens em *broadcast*

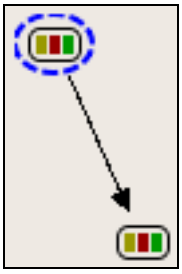

Figura A.3: Troca de mensagem entre sensores. O sentido da seta indica a direção do fluxo de dados durante o envio/recebimento.

0:0:6.94	3	Getting link to 0 with gain -114
0:0:6.94	2	Received message from node 3. Type = 112; Length = 7; Data = 0 0 80 ff ff 0 0
0:0:6.94	1	Received message from node 3. Type = 112; Length = 7; Data = 0 0 80 ff ff 0 0
0:0:7.0	0	Led 1 turned on
0:0:7.2	2	Broadcasting packet. Type = 112; Length = 7; Data = 0 0 80 ff ff 0 0

Figura A.4: Mensagem exibida na área de mensagens. A primeira coluna indica o tempo de simulação da ocorrência do evento, a segunda coluna tem o identificador do sensor que originou a ação e a terceira coluna apresenta uma mensagem explicativa da ação. A cor da linha identifica os diferentes tipos de mensagem.

Integração com o TOSSIM

O passo a passo da integração é mostrado usando a aplicação Blink que periodicamente altera o estado dos LEDs do sensor.

Copie para ele os arquivos disponíveis no diretório da aplicação Blink do TinyOS (e.g., /opt/tinyos-2.1.0/apps/Blink) para dentro de um novo diretório chamado "Blink". Tente compilar essa aplicação com o comando "make micaz sim-sf". Veja que não foi possível compilar a aplicação pela ausência do componente SerialActiveMessageC. A mensagem de erro foi:

```
(1)  In component `SerialActiveMessageC':
(2)  /opt/tinyos-2.1.0/tos/lib/tossim/sf/sim/SerialActiveMessageC.nc:
     In function `startDone.runTask':
(3)  /opt/tinyos-
     2.1.0/tos/lib/tossim/sf/sim/SerialActiveMessageC.nc:74:
     SplitControl.startDone not connected
(4)  /opt/tinyos-2.1.0/tos/lib/tossim/sf/sim/SerialActiveMessageC.nc:
     In function `stopDone.runTask':
(5)  /opt/tinyos-
     2.1.0/tos/lib/tossim/sf/sim/SerialActiveMessageC.nc:75:
     SplitControl.stopDone not connected
(6)  make: *** [sim-exe] Error 1
```

Para resolver este problema, é preciso que a aplicação use o componente SerialActiveMessageC implementando os commands startDone e stopDone. Eles não são usados dentro da aplicação, mas é preciso para a compilação voltada para uso do *SerialForwarder*. As linhas O arquivo de configuração BlinkAppC.nc deve ser atualizado de forma a ficar de acordo com o seguinte código:

```
(1)  configuration BlinkAppC {
(2)  }
(3)  implementation {
(4)    components MainC, BlinkC, LedsC;
(5)    components new TimerMilliC() as Timer0;
(6)    components new TimerMilliC() as Timer1;
(7)    components new TimerMilliC() as Timer2;
(8)    components SerialActiveMessageC as Serial;
(9)
(10)   BlinkC -> MainC.Boot;
(11)   BlinkC.Timer0 -> Timer0;
(12)   BlinkC.Timer1 -> Timer1;
(13)   BlinkC.Timer2 -> Timer2;
(14)   BlinkC.Leds -> LedsC;
(15)
```

```
(16)   BlinkC.SerialControl -> Serial;
(17) }
```

Foram adicionadas as linhas (8) e (16) no BlinkAppC. Em seguida, o arquivo de módulo BlinkC.nc teve as linhas (7)(33)(34) adicionadas, ficando da seguinte forma:

```
(1)  module BlinkC {
(2)    uses interface Timer<TMilli> as Timer0;
(3)    uses interface Timer<TMilli> as Timer1;
(4)    uses interface Timer<TMilli> as Timer2;
(5)    uses interface Leds;
(6)    uses interface Boot;
(7)    uses interface SplitControl as SerialControl;
(8)  }
(9)  implementation
(10) {
(11)   event void Boot.booted()
(12)   {
(13)     call Timer0.startPeriodic( 250 );
(14)     call Timer1.startPeriodic( 500 );
(15)     call Timer2.startPeriodic( 1000 );
(16)   }
(17)
(18)   event void Timer0.fired()
(19)   {
(20)     call Leds.led0Toggle();
(21)   }
(22)
(23)   event void Timer1.fired()
(24)   {
(25)     call Leds.led1Toggle();
(26)   }
(27)
(28)   event void Timer2.fired()
(29)   {
(30)     call Leds.led2Toggle();
(31)   }
(32)
(33)   event void SerialControl.startDone(error_t error) {}
(34)   event void SerialControl.stopDone(error_t error) {}
(35) }
```

Agora é a aplicação pode ser compilada usando o comando “make micaz sim-sf”.

Coloque os arquivos do pacote de distribuição do TOSViz no diretório da aplicação Blink. Abra o arquivo “main.c” e atualize a quantidade total de sensores da rede

na variável local "nodes" declarada dentro da função "main()". Compile os arquivos do TOSViz usando "make –f Makefile.Driver".

Inicialize o TOSViz com o comando "java –jar tosviz/TosViz.jar". A ferramenta deve inicializar com a opção de início de visualização desabilitada.

Em seguida, execute o comando "./main". Perceba que o TOSSIM começa a enviar mensagens pelo *SerialForwarder* e o TOSViz habilitou o botão para dar início a visualização da simulação. Pressione esse botão dar início a visualização. Neste momento, todos os sensores devem estar acendendo e apagando seus LEDs.

É possível realizar a visualização de qualquer aplicação simulada no TOSSIM realizando os passos descritos aqui nesta seção.

Funcionalidades

Alterando a topologia de visualização:

1. Iniciar a visualização;
2. Clicar e segurar o botão esquerdo do mouse sobre o sensor;
3. Arrastá-lo ao local desejado dentro do painel de simulação.

Exportando a topologia de visualização:

1. Alterar a topologia ou iniciar a visualização;
2. Selecionar "*Export Current Topology*" no menu "*File*";
3. Escolher nome e diretório para o arquivo a ser exportado;
4. Salvar o arquivo.

Importando a topologia de visualização:

1. Antes de ser iniciada a simulação, selecionar "*Import Topology*" no menu "*File*";
2. Escolher o arquivo a ser importado;
3. Selecionar a botão para abrir o arquivo selecionado.

Filtrando as mensagens:

1. Na área de mensagens, apertar o botão "*Filter...*";
2. Escolher o filtro;
3. Apertar o botão "*Ok*".

Visualizando as informações do sensor:

1. Na área de informações da simulação, selecionar um dos sensores listados;
2. Escolher a aba "*Topology*" ou "*Statistics*".

_____

Nelson Souto Rosa (Orientador)

_____

Rilter Tavares do Nascimento (Aluno)