



JESIMAR DA SILVA ARANTES

**MODELAGEM E IMPLEMENTAÇÃO DE UM
VISUALIZADOR PARA SIMULAÇÕES
COMPUTACIONAIS DE REDES DE
SENSORES SEM FIO**

LAVRAS – MG

2013

JESIMAR DA SILVA ARANTES

**MODELAGEM E IMPLEMENTAÇÃO DE UM VISUALIZADOR PARA
SIMULAÇÕES COMPUTACIONAIS DE REDES DE SENSORES SEM
FIO**

Monografia apresentada ao Colegiado do Curso de
Sistemas de Informação, para a obtenção do título
de Bacharel em Sistemas de Informação.

Orientador

Prof. Dr. Tales Heimfarth

LAVRAS – MG

2013

JESIMAR DA SILVA ARANTES

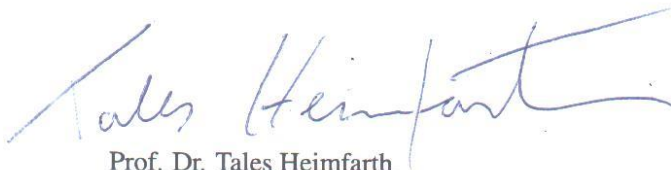
**MODELAGEM E IMPLEMENTAÇÃO DE UM VISUALIZADOR PARA
SIMULAÇÕES COMPUTACIONAIS DE REDES DE SENSORES SEM
FIO**

Monografia apresentada ao Colegiado do Curso de
Sistemas de Informação, para a obtenção do título
de Bacharel em Sistemas de Informação.

APROVADA em 30 de Agosto de 2013.

Prof. Dr. Raphael Winckler de Bettio UFLA


Prof. Dr. João Carlos Giacomini UFLA



Prof. Dr. Tales Heimfarth

(Orientador)

LAVRAS – MG

2013

À minha família!

*Meu pai Jésus Ferreira Arantes e minha mãe Sirley Ramos Arantes
que estiveram sempre presentes, me apoiando, me incentivando e me
ajudando nos momentos em que mais precisei.*

*Meu irmão Márcio da Silva Arantes pela amizade e ajuda,
e por saber que sempre terei com quem contar.*

*A minha tia Sirlaine Maria da Silva Freitas que me
mostrou a importância dos estudos.*

DEDICO

AGRADECIMENTOS

À Universidade Federal de Lavras (UFLA) e ao Departamento de Ciência da Computação (DCC), por me proporcionar um ambiente de estudo adequado à minha formação.

Aos professores da UFLA em especial aos professores do Departamento de Ciência da Computação, pelos ensinamentos transmitidos.

Ao professor Tales Heimfarth pela orientação, paciência, amizade e seus ensinamentos que foram de grande relevância para a realização deste trabalho e meu crescimento profissional. E a todos os integrantes do GRUBi que contribuíram e auxiliaram nas minhas pesquisas.

Aos meus colegas de apartamento do Alojamento Estudantil da UFLA que me fizeram crescer durante a minha jornada em Lavras. Aos colegas de turma que estiveram sempre presentes durante os meus estudos.

Agradeço sobretudo aos meus pais, Jésus Ferreira Arantes e Sirley Ramos Arantes, pelo amor e carinho. Ao meu irmão Márcio da Silva Arantes pela ajuda constante e amizade.

Muito obrigado por tudo!!!

EPIGRAFE

Tenho a impressão de ter sido uma criança brincando na praia, contente em achar aqui e ali, uma pedra mais lisa ou uma concha mais bonita, mas tendo sempre diante de mim, ainda por descobrir, o imenso oceano da verdade.

Isaac Newton

Estamos na situação de uma criança que entra numa grande biblioteca onde encontra muitos livros em muitas línguas diferentes. Ela sabe que alguém teve que escrever esses livros, mas não sabe como e não entende as linguagens em que estão escritos. A criança suspeita que existirá uma ordem no arranjo dos livros, mas não sabe qual é. Esta parece-me ser a atitude mais inteligente do ser humano perante Deus. Vemos um Universo que se estrutura e se move maravilhosamente mediante certas leis, mas mal entendemos essas leis. As nossas mentes limitadas não conseguem compreender integralmente a força que move as constelações.

Albert Einstein

Se não puder se destacar pelo talento, vença pelo esforço.

Dave Weinbaum

SUMÁRIO

1	INTRODUÇÃO	1
1.1	Contextualização	1
1.2	Motivação e Justificativa	3
1.3	Objetivos	4
1.3.1	Objetivo Geral	4
1.3.2	Objetivos Específicos	4
1.4	Estrutura do Trabalho	6
2	REFERENCIAL TEÓRICO	7
2.1	Redes de Sensores Sem Fios	7
2.2	Framework GrubiX	8
2.3	Visualizadores de RSSF	12
2.3.1	Visual ShoX	15
2.3.2	Nam	17
2.3.3	OMNeT++	18
2.3.4	TinyViz	19
2.3.5	MoteView	20
2.4	Modelo de Desenvolvimento de Software	22
2.4.1	Ciclo de Vida do Projeto	22
3	METODOLOGIA	25

3.1	Classificação do Trabalho	25
3.2	Desenvolvimento do Trabalho	26
4	DESENVOLVIMENTO DO TRABALHO	28
4.1	Visual GrubiX	28
4.1.1	Levantamento de Requisitos	28
4.1.2	Estrutura do Arquivo de Log	30
4.1.3	Arquitetura em Camadas	32
4.1.4	Diagrama de Pacotes	33
4.1.5	Diagrama de Classes	35
4.1.6	Arquitetura MVC	36
4.1.7	Algoritmo que Gerencia a Simulação	37
4.1.8	Principais Recursos Desenvolvidos	39
4.1.8.1	Mapping	40
4.1.8.2	Grafo de Conectividade	40
4.1.8.3	Linhas de Movimentação	42
5	RESULTADOS E DISCUSSÕES	43
6	CONCLUSÃO E TRABALHOS FUTUROS	48
	REFERÊNCIAS	50
A	ANEXO	53

LISTA DE FIGURAS

2.1	Componentes de um nó sensor e nó sensor IRIS. Fonte: Adaptado de (LOUREIRO <i>et al.</i> , 2003).	7
2.2	Diagrama de classes do <i>framework GrubiX</i> . Fonte: (LESSMANN; HEIMFARTH; JANACIK, 2008).	10
2.3	Etapas envolvidas no uso do simulador <i>GrubiX</i>	11
2.4	Funcionamento dos visualizadores de RSSF real. Fonte: Adaptado de (BUSCHMANN <i>et al.</i> , 2005).	14
2.5	Arquitetura MVC do <i>Visual ShoX</i> . Fonte: (LESSMANN; HEIMFARTH, 2008).	15
2.6	Interface gráfica do <i>Visual ShoX</i>	16
2.7	Interface gráfica do visualizador <i>nam</i>	18
2.8	Interface gráfica do visualizador do <i>OMNeT++</i>	19
2.9	Interface gráfica do visualizador <i>TinyViz</i>	20
2.10	Interface gráfica do programa <i>MoteView</i>	21
2.11	Hierarquia dos diagramas UML. Fonte: Adaptado de (BOOCH; JACOBSON; RUMBAUGH, 2006).	23
3.1	Classificação dos tipos de pesquisa. Fonte: (JUNG, 2004).	25
4.1	Estrutura do arquivos XML gerado pelo simulador.	31
4.2	Arquitetura em camadas do visualizador.	32
4.3	Diagrama de pacotes do <i>Visual GrubiX</i>	34

4.4	Diagramas de classes dos pacotes <i>Control</i> e <i>Structure</i>	36
4.5	Diagramas de classes dos pacotes <i>WINDOW</i> e <i>PAINT</i>	36
4.6	Arquitetura MVC do <i>Visual GrubiX</i>	37
4.7	Exemplo de grafo de conectividade da rede.	41
4.8	Exemplo de linhas de movimentação da rede.	41
5.1	Tela inicial do <i>Visual GrubiX</i>	44
5.2	Tela de simulação do <i>Visual GrubiX</i>	44
5.3	Tela de simulação do <i>Visual GrubiX</i>	45
B.1	Diagrama de classes da camada <i>Controller</i>	55
B.2	Diagrama de classes da camada <i>View</i>	56
B.3	Diagrama de classes completo do <i>Visual GrubiX</i>	57

LISTA DE TABELAS

5.1	Comparação do <i>Visual GrubiX</i> e do <i>Visual ShoX</i>	46
5.2	Comparação geral dos visualizadores <i>Visual GrubiX</i> , <i>Visual ShoX</i> , <i>nam</i> e <i>OMNeT++</i>	46

RESUMO

Este trabalho apresenta a modelagem e implementação do Visual GrubiX, uma ferramenta de visualização de rede de sensores para o simulador GrubiX. A tarefa de visualizar uma simulação complexa de rede traz vários desafios, como por exemplo a forma de expressar a troca de pacotes, a visualização dos diferentes tipos de pacotes etc. Na literatura, diversas ferramentas de visualização podem ser encontradas, no entanto, na sua maioria são específicas para um determinado simulador e faltam vários recursos de visualização. O Visual GrubiX supera essas deficiências, utilizando um arquivo XML como entrada para a visualização e permitindo um visual rico e uma depuração de simulações de rede. Além disso, um esquema de mapeamento permite ao usuário especificar quais elementos gráficos que representam um determinado aspecto de simulação.

Palavras-Chave: Redes de Sensores Sem Fios; Simulação Computacional; Visualização de Simulações em RSSF.

ABSTRACT

This work presents the design and implementation Visual GrubiX, a sensor network visualisation tool for the GrubiX simulator. The task of visualizing a complex network simulation brings several challenges, e.g. how to express the packet exchange, how to visualize different packet types, etc. In the literature, several visualization tools can be found, nevertheless they are mostly specific for a certain simulator and lack of several visualization features. The Visual Grubix overcomes these shortcomings, using a XML file as input for visualization and enabling a visual-rich debugging of network simulations. Moreover, a mapping scheme allows the user to specify which graphical elements will represent a given simulation aspect.

Keywords: Wireless Sensor Networks, Computer Simulation; View Simulations WSN.

1 INTRODUÇÃO

Este trabalho tem como foco principal apresentar a modelagem e implementação de um software de visualização de simulações em Redes de Sensores Sem Fio (RSSF). Este programa exhibe, através de uma interface gráfica amigável, as simulações de protocolos e aplicações feitas no *framework GrubiX*.

1.1 Contextualização

As Redes de Sensores Sem Fio (RSSFs) são um conjunto de nós sensores que se comunicam usando ligações sem fios. Esses nós sensores são constituídos de um comunicador sem fio, uma fonte de energia, uma unidade de sensoriamento, memória e um processador (LOUREIRO *et al.*, 2003). Geralmente, eles trabalham na realização de um objetivo comum, como monitoramento ambiental, rastreamento, coordenação e processamento em diferentes contextos. Além disso, uma vez que os nós sem fios são pequenos e têm recursos limitados, geralmente não é viável implementar algoritmos que exigem um grande poder de processamento ou muito espaço de memória. Tudo isso faz com que o projeto de protocolos para RSSF seja uma tarefa desafiadora. Combinado com o fato de que essas redes podem ser utilizadas para missões críticas, como detecção de incêndio florestal, é inevitável ter que efetuar exaustivamente testes com os protocolos.

Há duas formas de se fazer esses testes: programar os nós sensores e colocá-los para funcionar em uma rede real, ou utilizar um programa de simulação de RSSF. A primeira forma pode produzir resultados mais precisos entretanto há várias desvantagens como alto custo de hardware, dificuldade para testar em um grande número de nós com movimentos realistas, etc. Sendo assim, as implementações reais geralmente serão uma opção viável apenas para um menor número de

nós e durante as fases posteriores de implementação quando mudanças significativas na base do código não são esperadas. Já a segunda opção evita os problemas das implementações reais descritos acima. Por outro lado, a simulação pode fornecer uma previsão da realidade apenas numa extensão limitada, o que implica que os resultados de simulações não são tão precisos quanto implementações reais. Ainda assim, as simulações são utilizadas em muitas situações em que uma precisão limitada pode ser tolerada.

Mesmo que RSSFs sejam formadas por componentes de baixo custo (nós sensores), é relevante em etapas anteriores à instalação da rede e testes em campo, a utilização de simulação computacional para visualizar o comportamento funcional da rede (algoritmos desenvolvidos). Isso é relevante em redes que apresentem grande quantidade de nós, onde a visualização dos resultados em testes reais torna-se complicada. Ou seja, toda a modelagem inicial do problema deve ser feita utilizando um simulador específico para facilitar o trabalho.

Além disso, simular sistemas reais torna-se uma tarefa onerosa quando seus elementos (nós e sensores) apresentam partes que se diferenciam em termos de tecnologia de construção (sistemas heterogêneos). Portanto, é de grande importância a utilização de plataformas de simulação computacional que supram essas características (MELLO; CAIMI, 2008).

Existem, no mercado, vários *frameworks* para simulações de redes sem fio. Esses *frameworks*, pelo fato de atuarem apenas em ambientes virtuais (simulados) acabam por garantir um grande poder de expressividade e facilidade na confecção dos algoritmos. Tais ferramentas permitem que dados gerados ao longo da simulação sejam armazenados em arquivos, denominados *log* de saída. Entretanto tais dados ficam em arquivos de textos e são de difícil interpretação pelo projetista da rede. Sendo assim se faz necessário o uso de ferramentas de visualizações das

simulações executadas de maneira gráfica e intuitiva. Esses *frameworks* têm em geral seus próprios visualizadores.

Este trabalho tem como foco simulações computacionais em RSSFs e não em implementações reais de redes de sensores. O simulador utilizado é o *framework GrubiX* (GRUBIX, 2013) o qual é baseado no *ShoX (scalable ad hoc network simulator)* (SHOX, 2012) que é utilizado em redes móveis *ad hoc* (LESSMANN; HEIMFARTH; JANACIK, 2008). Sendo assim o *GrubiX* é utilizado em redes móveis *ad hoc*, mas também pode ser utilizado em RSSFs.

O presente trabalho apresenta o desenvolvimento de uma ferramenta de visualização para o *framework GrubiX*. O software proposto, que será chamado neste trabalho apenas por *Visual GrubiX*, suporta também visualizar aplicações desenvolvidas no *ShoX*, já que os dois são compatíveis. O *ShoX* também possui um visualizador (LESSMANN; HEIMFARTH, 2008) que antes do desenvolvimento deste trabalho era utilizado juntamente com o *GrubiX*. Esse visualizador do *ShoX* será chamado aqui de *Visual ShoX*.

1.2 Motivação e Justificativa

A grande motivação para o desenvolvimento de um software visualizador de Redes de Sensores Sem Fio para o *GrubiX* é o fato do atual visualizador, o *Visual ShoX*, apresentar algumas deficiências que serão mostradas neste trabalho. Por este motivo, foi desenvolvido um visualizador de simulações em RSSF que dê suporte ao *GrubiX* e que seja amigável, fácil de usar e tenha um código com alto grau de manutenibilidade.

Uma motivação que impulsiona este trabalho é o fato da área de RSSF ser uma tecnologia chave para o futuro. Em Setembro de 1999, a revista *Business Week Magazine* anunciou as redes de sensores como uma das 21 mais importantes

tecnologias para o século XXI. Além destas motivações existe o fato de não existirem muitos softwares de simulação especializados em RSSF, assim fazer um bom visualizador para o *GrubiX* irá contribuir de maneira significativa para a área que esta em crescimento constante.

A justificativa para se trabalhar com simulações computacionais se dá pelo fato de trabalhos com sistemas reais serem difíceis de se implementar e os testes e equipamentos serem onerosas. Outra justificativa é o fato de que trabalhar com simulações computacionais serem mais fáceis de extrapolar limitações de equipamentos físicos, como por exemplo fazer experiências com milhares de nós o que fica inviável de fazer com testes reais.

1.3 Objetivos

Os objetivos presentes neste trabalho são apresentados a seguir.

1.3.1 Objetivo Geral

Esse projeto tem como objetivo geral a modelagem e implementação de um software para visualização de simulações computacionais. Este visualizador deve fazer a leitura de um arquivo de *log* (arquivo de registro de resultados) gerado pelo *GrubiX*, processar o conteúdo no formato textual deste arquivo e o transformar em conteúdo visual e de fácil visualização, através de uma interface gráfica amigável.

1.3.2 Objetivos Específicos

1. Elaboração do Projeto:

- **Pesquisa em bibliografia sobre visualizadores em RSSF:** Fazer o levantamento de trabalhos relacionados ao apresentado.

- **Elaboração de um software visualizador de RSSF:** Elaborar o software para visualização de simulações em RSSF.
- **Comparação com outros softwares de visualização em RSSF:** Efetuar a comparação com outros softwares de RSSF existentes no mercado.

2. Desenvolvimento do Projeto:

- **Levantamento de Requisitos:** O levantamento de requisitos para a confecção do programa é uma das etapas mais importantes do projeto. Esta etapa é responsável por listar todos os requisitos funcionais e não funcionais do software norteando e delimitando assim o escopo do mesmo.
- **Diagramas de Classes:** A construção dos diagramas de classe tem como objetivo preparar o terreno para a implementação, já que esses mostram a hierarquia presente no projeto e como as classes se relacionam entre si.
- **Diagramas de Pacotes:** Assim como a construção dos diagramas de classes este diagrama também prepara o terreno para a implementação do software. Este diagrama tem como principal objetivo mostrar a hierarquia das classes e procura agrupá-las de maneira organizada por função ou estrutura os quais manipulam.
- **Definição do Modelo de Desenvolvimento de Software:** Fazer um estudo sobre os modelos de desenvolvimento de software e verificar qual a melhor opção a se aplicar neste trabalho.
- **Implementação:** Efetuar a codificação do projeto, obedecendo aos diagramas propostos (classes e pacotes). Efetuar a leitura do arquivo de *log* transformar o mesmo em conteúdo visual.

1.4 Estrutura do Trabalho

Este trabalho está organizado da seguinte forma. O Capítulo 2 apresenta o referencial teórico levantado durante este trabalho. O Capítulo 3 mostra a metodologia de trabalho utilizada na modelagem e implementação do visualizador proposto. O Capítulo 4 apresenta o desenvolvimento do trabalho como o levantamento de requisitos e a elaboração da arquitetura, entre outras etapas. O Capítulo 5 apresenta resultados e discussões do trabalho corrente. No Capítulo 6 são apresentados as conclusões.

2 REFERENCIAL TEÓRICO

2.1 Redes de Sensores Sem Fios

Devido aos avanços da tecnologia, Redes de Sensores Sem Fio (RSSFs) têm apresentado uma ampla variedade de aplicações, que podem ser comerciais, científicas e/ou militares. Uma RSSF é composta por um conjunto de elementos, chamados nós sensores. O motivo da sua utilização é o baixo custo de seus elementos, pequeno tamanho e seu uso simplificado. O nó sensor é um sistema embarcado simples composto por um transceptor (rádio) com antena integrada, uma bateria, um processador básico, uma pequena quantidade de memória e vários sensores. A Figura 2.1 mostra um diagrama de blocos de um nó sensor com seus componentes básicos.

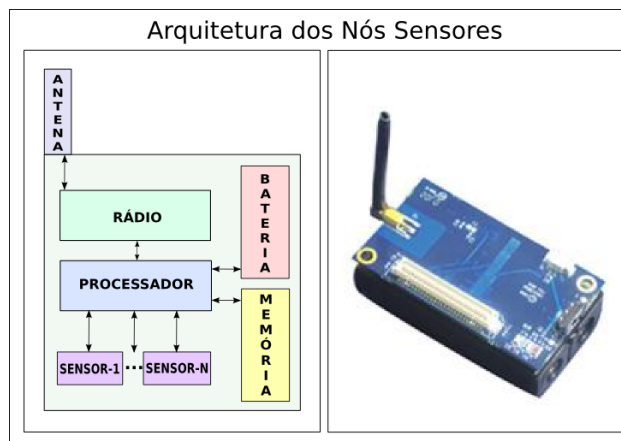


Figura 2.1: Componentes de um nó sensor e nó sensor IRIS. Fonte: Adaptado de (LOUREIRO *et al.*, 2003).

A limitação de hardware faz com que os nós de uma rede sejam colocados em forma de *clusters* ou grupos cooperantes para realização de tarefas complexas de sensoriamento. Isso também é feito, porque RSSFs quando colocadas dessa forma apresentam características desejáveis, tais como: robustez, adaptabilidade, auto-

nomia e escalabilidade, que estão presentes na maioria de sistemas massivamente distribuídos.

Para que se realizem as tarefas, a densidade da rede também é um fator avaliado. A maior quantidade de nós na rede implica em uma melhor qualidade de sensoriamento, assim como uma grande quantidade de bons enlaces de comunicação sem fio. Segundo (SILVA, 2006), esse tipo de rede é utilizado como uma boa alternativa ao sensoriamento remoto ou a outros métodos tradicionais, justo que permite com sua estreita relação com o meio físico, um sensoriamento detalhado.

Mesmo que RSSFs sejam formadas por componentes de baixo custo, é relevante em etapas anteriores à instalação da rede e testes em campo, a utilização de simulação computacional para visualizar o comportamento funcional da rede (algoritmos desenvolvidos). Isso é primordial em redes que apresentam de centenas a milhares de nós, onde a visualização dos resultados em testes reais torna-se complicada. Ou seja, toda a modelagem inicial deve ser feita utilizando um simulador específico à aplicação para facilitar o trabalho.

2.2 Framework GrubiX

O simulador *GrubiX* foi desenvolvido na Linguagem Java e permite facilmente criar aplicações ligadas a RSSFs. Este programa segue o paradigma da programação orientada a objetos. Para se desenvolver uma aplicação neste simulador deve-se utilizar as classes mais genéricas e funcionalidades já implementadas neste *framework*. Após concluída e executada a aplicação, será gerado um arquivo de saída XML. Este é o arquivo que vai ser utilizado para se visualizar a aplicação desenvolvida no *Visual GrubiX*.

O *GrubiX* é um *framework* para simulações computacionais em RSSFs. Este software, como a maioria dos simuladores de rede populares, é orientado a eventos

discretos. Isto significa que existe uma fila de eventos globais em que todos os eventos de rede (movimentos de nó, pacotes, mensagens internas, temporizadores, etc) são inseridos. Cada evento tem um identificador (ID) único e um relógio que especifica o seu tempo de entrega, ou seja, o momento em que ele deve ser retirado da fila e entregue ao seu destinatário, onde é processado. O tempo de entrega também define a ordem segundo a qual os eventos são armazenados na fila de eventos.

A entidade central do *GrubiX* é a `SIMULATIONMANAGER`, como mostra a Figura 2.2. Ela gerencia a fila de eventos e sempre busca o primeiro evento da fila e o entrega ao nó especificado. Enquanto isso, ela modifica o tempo de simulação atual para o armazenado no evento corrente. Dessa forma, é possível simular paralelismo verdadeiro.

Todos os conceitos conhecidos a partir do domínio de redes sem fio como camadas do modelo OSI, pacotes, modelos de mobilidade, e outros, são modelados como classes abstratas, como mostra a Figura 2.2. Nesta figura é mostrado o diagrama de classes simplificado do *GrubiX*. Criando subclasses dessas classes pode-se definir novas implementações, para RSSF. Isto torna muito fácil e seguro a definição de modelos de mobilidade próprios ou protocolos.

Os pacotes são nada além de eventos especiais. Quando um pacote é criado em uma determinada camada, por exemplo a camada de aplicação, é enviado para baixo na pilha de camadas, até atingir a camada física. Abaixo da camada física, existe uma camada especial artificial chamado `AIRMODULE`. Esta camada gerencia o estado do rádio de um nó sensor (enviar, receber, ocioso, dormindo, desligado) e também é responsável por manter o controle de possíveis interferências de outros nós.

Para efeitos do modelo de propagação do sinal de rádio, uma classe abstrata chamada `PHYSICALMODEL` é fornecida. Dadas as posições geográficas de envio

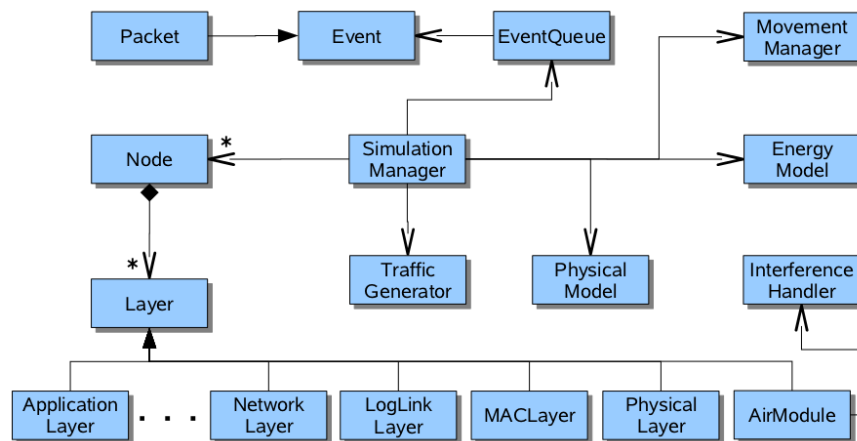


Figura 2.2: Diagrama de classes do *framework GrubiX*. Fonte: (LESSMANN; HEIMFARTH; JANACIK, 2008).

e recepção do nó, e a intensidade do sinal com o qual o remetente transmite o pacote, a implementações do PHYSICALMODEL calcula se o receptor pode ou não ser atingido pelo remetente. Se o receptor está a uma distância alcançável ou, pelo menos, de interferência, o PHYSICALMODEL retorna a intensidade do sinal recebido. Para mais detalhes consulte (LESSMANN; HEIMFARTH; JANACIK, 2008).

Um esquema geral do processo de funcionamento do *framework* pode ser visto na Figura 2.3. Conforme pode ser observado, primeiro se constrói uma aplicação que usa as classes do simulador *GrubiX*. Em seguida executa-se esta aplicação, gerando um arquivo de saída no formato XML. Este arquivo de *log* é gerado através de recursos presentes no simulador *GrubiX*. Este arquivo contém toda as informações provenientes da simulação da rede. O *Visual GrubiX* faz a leitura do arquivo gerado, processa internamente os dados deste arquivo e mostra suas informações em um formato visual. Cabe ao usuário o desenvolvimento da aplicação e o controle do software de visualização da simulação.

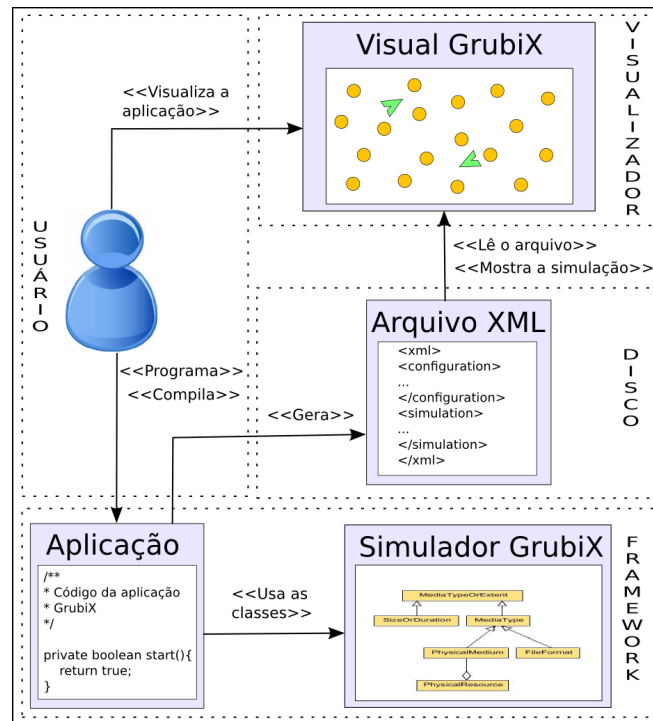


Figura 2.3: Etapas envolvidas no uso do simulador *GrubiX*.

O *GrubiX* foi modelado especificamente para RSSFs ao contrário da maioria dos outros simuladores que eram específicos para redes sem fios e foram estendidos para RSSFs, como é o caso do *OptNet*. As funcionalidades do *GrubiX* e suas características estão descritas abaixo e são baseadas em (LESSMANN; HEIMFARTH; JANACIK, 2008):

1. O *GrubiX* foi implementado de forma modularizada, portanto funcionalidades (protocolos) podem ser incluídas ao simulador de maneira flexível.
2. É um simulador baseado na linguagem de programação Java (para implementação) e na de marcação XML (para configuração e saídas exportadas).
3. A linguagem Java elimina questões de programação problemáticas, como falha de segmentação, alocação e desalocação de memória, que estão pre-

sentas nas linguagens C e C++. O XML permite o uso de ferramentas padronizadas para análise (*parsing*) de arquivos e tradução de arquivos para outros formatos.

4. Todo o conceito de camadas OSI (protocolos), pacotes, mobilidade, propagação de sinal e modelos de tráfego são definidos como super-classes abstratas. Para definir novos protocolos e outros modelos é suficiente que a sub-classe estenda a classe abstrata, a qual provê *frameworks* e um padrão de desenvolvimento de novos conceitos.
5. É um simulador orientado a eventos discretos. Para cada evento, que pode ser qualquer ação ou detecção de ações dentro da rede, geram-se outros eventos, controlados pelo projetista da aplicação. Assume-se também que eventos acontecem em qualquer instante durante a simulação da rede.
6. Permite seguir o paradigma orientado a objetos para desenvolvimento das aplicações.
7. O *GrubiX* possui suporte GUI (*Graphical User Interface*) para configuração de cenários e visualização da RSSF.
8. Durante a simulação, todo comportamento da rede de sensores é guardado em um arquivo de *log* para uma futura visualização e análise dos resultados.

2.3 Visualizadores de RSSF

Existem no mercado uma grande variedade de *frameworks* de RSSFs. A maioria destes softwares possuem uma ou mais ferramentas de visualização (PARBAT; DWIVEDI; VYAS, 2010). Os dados coletados a partir da RSSF geralmente são salvos usando alguma codificação em uma estação base central ou arquivo. Exis-

tem muitos programas que facilitam a visualização destas grandes quantidades de dados coletados como o *Visual ShoX*, *Nam*, *TinyViz*, *MoteView*, entre outros.

Depois de estudar e avaliar uma série de trabalhos de pesquisa, se deparou com a dificuldade de encontrar trabalhos sobre softwares de visualização de simulações em RSSF. A grande maioria dos trabalhos encontrados na área focam em simulação, protocolos de roteamento, arquiteturas, coleta de dados, mineração de dados do sensor, questões de segurança etc. Foram encontradas poucas pesquisas e literaturas disponíveis com foco em ferramentas de visualização de dados para RSSFs (PARBAT; DWIVEDI; VYAS, 2010). O principal objetivo desta seção é apresentar um levantamento detalhado sobre as ferramentas de visualização que são usadas para visualizar as informações coletadas nas RSSFs.

Alguns autores da área classificam os simuladores de RSSFs em duas classes de softwares. Na primeira classe tem-se os programas que fazem a visualização de implementações reais. Esta classe é chamada por alguns de emuladores. Sendo assim, o software de visualização captura as informações do *gateway*, o qual obteve as informações dos nós sensores ou do nó *sink*. O *sink* é o nó que faz a conexão da rede com uma estação concentradora de dados, como mostra a Figura 2.4 (nesta figura o *sink* é o último nó sensor antes da ligação com o *gateway*). Na outra classe os programas de simulação fazem toda a construção da rede virtual. Nesta classe tais softwares são chamados de simuladores. Assim o visualizador deste segundo tipo faz a captura dos dados simulados em um *framework* de RSSF que modelou todo o ambiente e a rede.

Os visualizadores da primeira classe precisam ser ágeis e filtrarem os dados rapidamente. Devem também fornecer uma avaliação significativa do estado da rede. Outras características que devem possuir são a habilidade de disponibilizarem os dados da rede de sensores para o utilizador e apresentar as informações da forma mais eficaz e funcional possível.

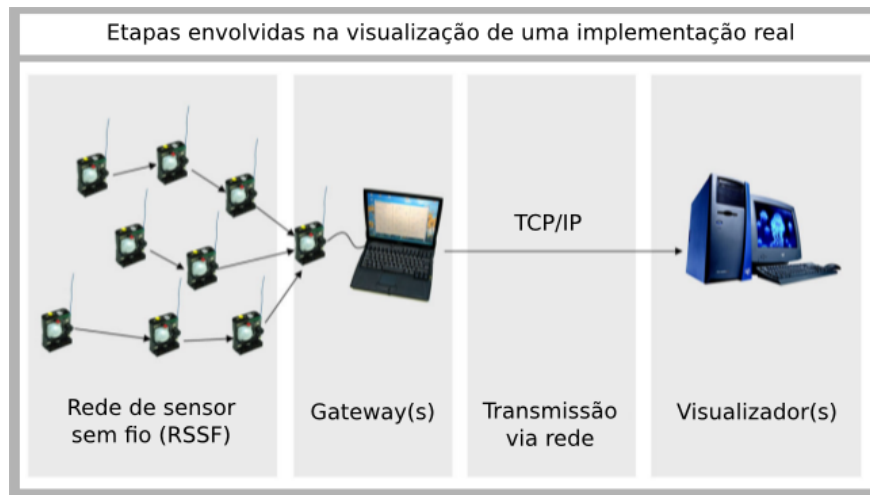


Figura 2.4: Funcionamento dos visualizadores de RSSF real. Fonte: Adaptado de (BUSCHMANN *et al.*, 2005).

A visualização de eventos de rede pode ser classificada em visualização *offline* e *online*. A visualização *online* caracteriza-se pela exibição dos eventos durante a execução da simulação. A visualização *offline*, os eventos são registrados em um arquivo de *log* e exibidos somente após o término da simulação. Ambas as formas de visualização possuem suas vantagens. A visualização *online* permite ver o comportamento da rede em curso imediatamente. Caso algum estado indesejado ocorra, o usuário poderá tomar uma decisão imediatamente, como a interrupção de uma simulação demorada. Por outro lado, a visualização *offline* permite retroceder a algum ponto no tempo anterior. Dessa forma, eventos específicos que chamaram a atenção do desenvolvedor, mas eram muito curtos para serem compreendidos plenamente podem ser convenientemente repetidos quantas vezes se desejar. Outra característica da visualização *offline* é a possibilidade de se mudar a velocidade da animação. Com a visualização *online*, o usuário teria que refazer toda a simulação novamente.

A seguir são apresentados alguns exemplos de visualizadores criados para Redes de Sensores Sem Fio.

2.3.1 Visual ShoX

O *Visual ShoX* é um visualizador de RSSF desenvolvido para o *framework ShoX*. O *Visual ShoX* é um visualizador *offline*. Durante a simulação, este simulador registra todos os eventos relevantes da rede em um arquivo de *log* no formato XML (*Extensible Markup Language*).

A Figura 2.5 apresenta uma visão geral dos componentes individuais do *Visual ShoX* e suas dependências. A arquitetura básica segue o paradigma *Model-View-Controller*. Primeiro, o arquivo de *log* é carregado em uma lista de eventos e configurações gerais da rede. A lista de eventos é uma coleção de eventos ordenados pelo tempo (LESSMANN; HEIMFARTH, 2008).

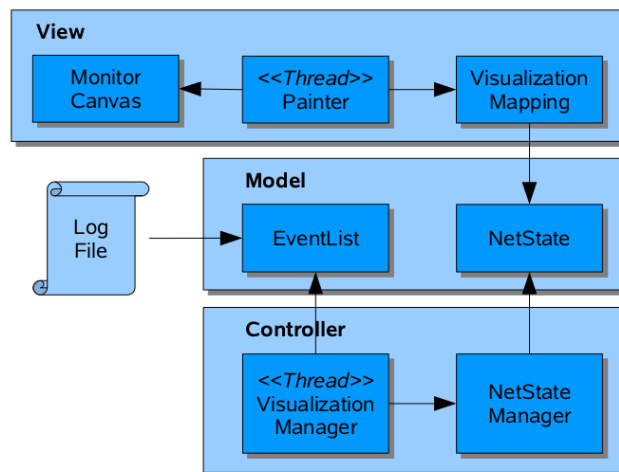


Figura 2.5: Arquitetura MVC do *Visual ShoX*. Fonte: (LESSMANN; HEIMFARTH, 2008).

Uma vez que o processo de visualização é iniciado, o `VISUALIZATIONMANAGER` repetidamente busca o evento atual do `EVENTLIST`. Com base no tempo de entrega, o `VISUALIZATIONMANAGER` calcula o tempo do mundo real correspondente, isto é, o tempo em que o evento deve ser visualizado de acordo com a velocidade que o utilizador selecionou para visualização. No momento da vi-

sualização pelo computador do evento corrente, o `VISUALIZATIONMANAGER` se comunica com o `NETSTATEMANAGER` para que o mesmo atualize o `NETSTATE` (LESSMANN; HEIMFARTH, 2008).

A Figura 2.6 mostra a interface deste software, o qual mostra uma simulação em execução. Esta interface é composta de três áreas. Abaixo estão localizados os botões de controle. É possível iniciar, pausar, continuar e reiniciar o processo de visualização da simulação. Além disso, há vários níveis de velocidade e *zoom*. Uma característica interessante é a possibilidade de capturar a animação em um vídeo *MPEG*. Ao pressionar o botão “`REC_ON`”, uma imagem da área de animação é tomada em intervalos fixos (LESSMANN; HEIMFARTH, 2008). As imagens individuais são então reunidos em um vídeo usando o utilitário *Transcode* (TRANSCODE, 2012).

A área central da tela mostra uma animação de uma simulação. Ela mostra os nós, os links e os pacotes. Na Figura 2.6, pode-se ver que um pacote está sendo enviado na parte inferior da área de animação (representado por um retângulo pequeno).

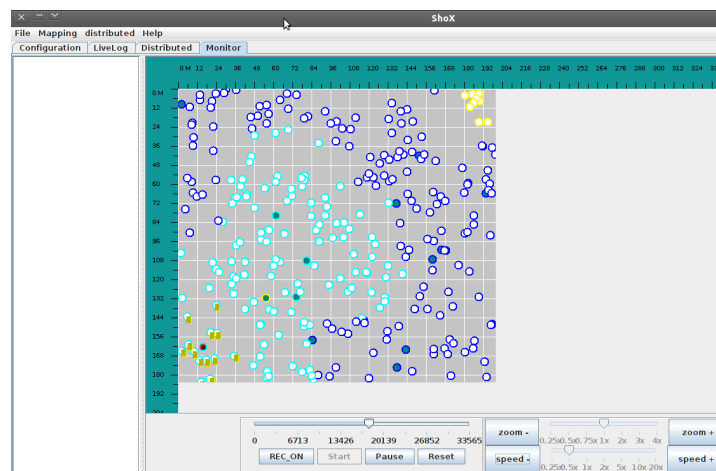


Figura 2.6: Interface gráfica do *Visual ShoX*.

No *Visual ShoX*, a aparência gráfica dos nós e ligações podem ser alteradas de forma flexível. Particularmente, a aparência pode ser usada para descrever determinado nó ou estados de ligação. Antes ou durante o processo de visualização, um mapeamento pode ser feito pelo utilizador na GUI entre os estados registrados e a representação gráfica pretendida. Dessa forma um nó pode ser mapeado para um rótulo que é visivelmente ligado a ele. Da mesma forma, um nó pode ter variadas cores, tamanho, espessura da borda e rótulos diferentes definidos no programa. Assim, uma variedade de informações de estado interessante podem ser exibidas de uma forma intuitiva para o usuário com facilidade. Por ser um visualizador *offline* o *Visual ShoX* possui todas as características das simulações *offline*, como voltar no tempo, selecionar uma parte do tempo específica, alterar a velocidade do processo de animação etc. Outras informações podem ser obtidas em (JANACIK *et al.*, 2010).

2.3.2 Nam

Uma ferramenta para simulações de rede é o Network Simulator (*ns-2*), e seu visualizador é conhecido como *nam*. Este, assim como o *Visual ShoX*, é um simulador *offline*. Isto implica que o simulador produz arquivos que são visualizados apenas após a simulação ter terminado. Neste visualizador os nós podem ter três formas que são: círculos, retângulos e hexágonos.

A interface deste visualizador é mostrada na Figura 2.7. Uma característica na forma de visualização dos nós é que as mesmas têm de ser definidas antes da simulação ser iniciada e não podem ser alterada em tempo de execução. Durante a execução, apenas a cor do nó pode ser alterada. Além disso, os nós podem ter rótulos arbitrários. Da mesma forma, as ligações podem mudar sua cor de forma dinâmica e podem ter rótulos. Assim, o conjunto de propriedades gráficas

que podem ser manipulados pelo usuário é bastante limitado em comparação com *OMNeT++* que será descrito a seguir.

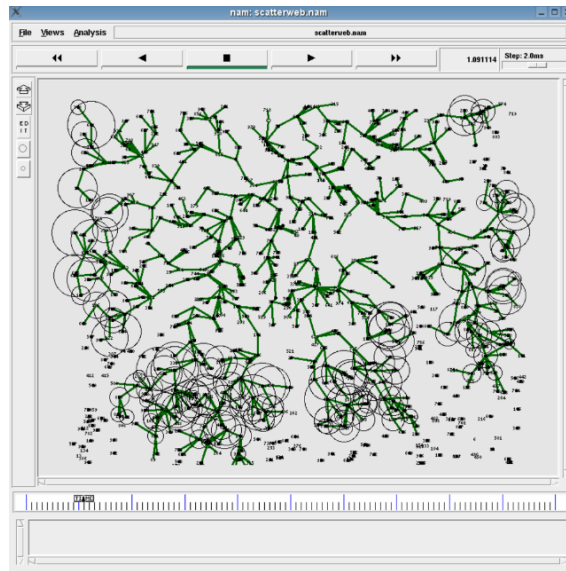


Figura 2.7: Interface gráfica do visualizador *nam*.

Pelo fato do *nam* ser um visualizador *offline*, este possui todas as características das simulações *offline*, como voltar num tempo anterior da simulação, selecionar uma parte específica do tempo da simulação e alterar a velocidade da animação.

2.3.3 OMNeT++

O *OMNeT++* (VARGA, 2012) é um exemplo de uma ferramenta de simulação com visualização *online* ao contrário dos visualizadores *Visual ShoX* e *nam*. Como consequência de ser *online*, não existe comando para retroceder a simulação. Por outro lado, o *OMNeT++* permite inspecionar as variáveis e estados de todos os objetos da rede de uma maneira muito detalhada. Os valores podem ainda ser

alterados em tempo de execução para avaliar o impacto de certas propriedades no curso da simulação.

A interface gráfica deste visualizador pode ser vista na Figura 2.8. Este simulador também permite definir a forma como os estados do nó ou do *link* são representados graficamente. É possível ajustar a cor, a espessura da borda do nó, forma do nó (retângulo ou elipse), tamanho, rótulo, e espessura.

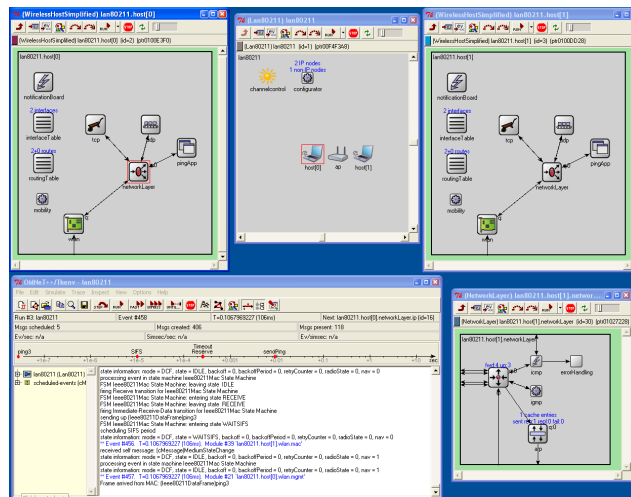


Figura 2.8: Interface gráfica do visualizador do OMNeT++.

2.3.4 TinyViz

O *TinyViz* é uma ferramenta de visualização de simulações para RSSF. Esta ferramenta é utilizada no *TOSSIM*, que foi um dos primeiros simuladores puramente orientados para redes de sensores (LEVIS *et al.*, 2003). A Figura 2.9 mostra a interface gráfica do visualizador *TinyViz*.

O visualização *TinyViz* é o mais genérico entre a maioria dos outros visualizadores, é também um visualizador *online*. Ele apresenta as leituras dos sensores,

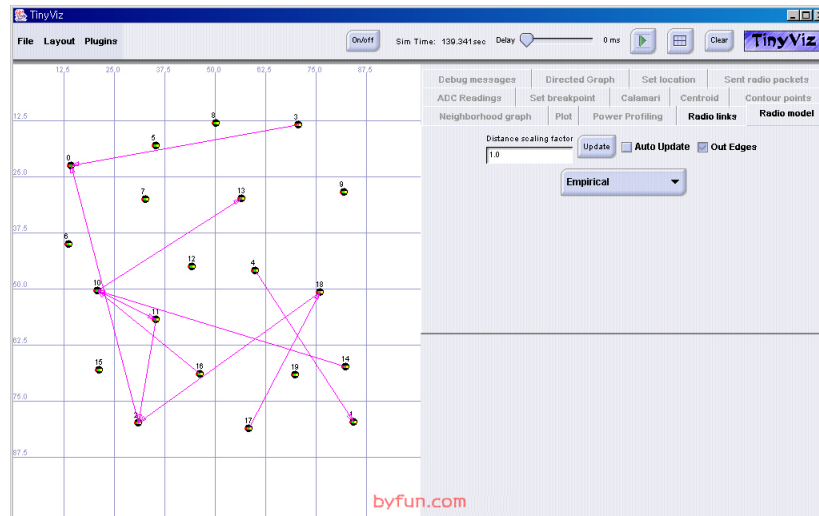


Figura 2.9: Interface gráfica do visualizador *TinyViz*.

estados dos LEDs, ligações entre os rádios e permite a interação direta com simulações do *TOSSIM*.

A arquitetura de *TinyViz* permite adicionar funcionalidades de visualização do aplicativo específico. Esta funcionalidade inclui as operações de desenho especializadas, de subscrição e de reação a eventos e fornecer *feedback* para o simulador *TOSSIM*. O *TinyViz* possui a capacidade de executar a visualização em tempo real da simulação.

O *TinyViz* foi desenvolvido para a versão 1 do *TinyS*, não foi feito para a versão 2 e não é mais usado.

2.3.5 MoteView

O software de monitoramento *MoteView* foi desenvolvido pela empresa *Crossbow* para visualizar dados de RSSF (CROSSBOW, 2007). Este visualizador é um visualizador de rede real. Este programa é projetado para ser uma interface da “Camada Cliente” entre o usuário e a RSSF implantada. *MoteView* oferece

aos usuários as ferramentas para simplificar a implantação e o monitoramento da RSSF. Também torna mais fácil se conectar a um banco de dados, analisar e fazer as leituras dos sensores através dos gráficos. A arquitetura do *MoteView* é baseada em módulos bem definidos e divididos.

A Figura 2.10 mostra a interface gráfica do *MoteView*. De acordo com (PAR-BAT; DWIVEDI; VYAS, 2010) a tela principal da interface do usuário consiste de sete abas das quais quatro são dados, comandos, gráficos e topologia. Na aba dados são exibidas as leituras mais recentes dos sensores que foram recebidas de cada nó na rede. Na aba gráficos o modo de exibição fornece a capacidade de gerar gráficos da leitura do sensor em função do tempo para um dado conjunto de nós sensores. Na aba topologia é mostrado um mapa topológico da rede de sensores. Na aba comando o usuário tem a capacidade de alterar os parâmetros dos nós da rede de sensores sem fio.

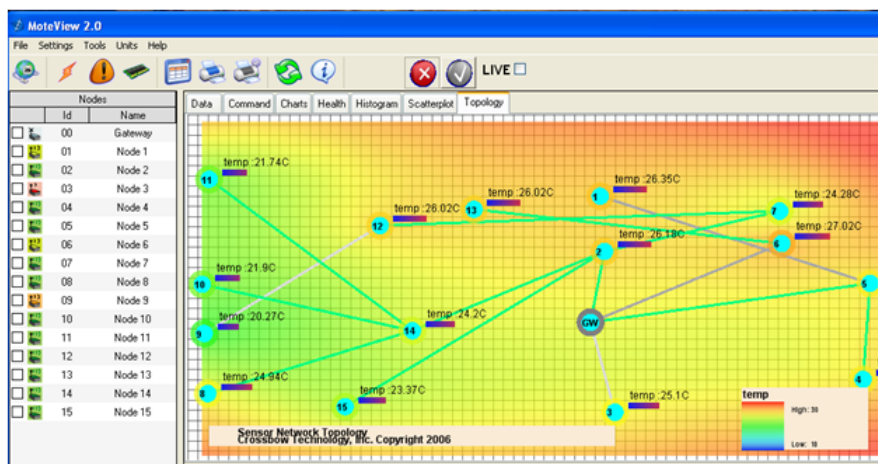


Figura 2.10: Interface gráfica do programa *MoteView*.

O *MoteView* apresenta muitas características topológicas e de visualização das estatísticas da rede, bem como o registro de leituras de sensores, além da visualização dos dados registrados. A função estatística inclui, por exemplo, o rendimento de pacotes de dados fim a fim. Ele permite consultar a rede de sensores para co-

letar dados do banco de dados. Os usuários podem habilitar opções para desenhar ligações entre os nós e especificar se o *Gateway* tem um sensor nele para a visualização do gradiente. O *MoteView* possui um gerenciador de alertas e permite ao usuário definir situação de alertas com base em quaisquer dados de sensores de qualquer nó sensor. Possui como limitação o fato de executar somente nos sistemas operacionais da família Windows.

2.4 Modelo de Desenvolvimento de Software

O desenvolvimento de um projeto necessita de planejamento sobre todas as suas etapas. Para isto este trabalho seguiu algumas etapas descritas pelo RUP - *Rational Unified Process*. O RUP é um processo proprietário de Engenharia de Software criado pela *Rational Software Corporation*, adquirida pela IBM, fornecendo técnicas a serem seguidas pelos membros da equipe de desenvolvimento de software com o objetivo de aumentar a sua produtividade no processo de desenvolvimento. O RUP faz uso dos diagramas UML (*Unified Modeling Language*) (BOOCH; JACOBSON; RUMBAUGH, 2006). Estes diagramas possuem uma notação universalmente difundida e auxiliam a projetar um software e a compreender as etapas de desenvolvimento mais claramente.

A Figura 2.11 mostra a hierarquia entre os mais diversos diagramas definidas pela UML. O conhecimento dos principais tipos de diagramas UML é de extrema importância para o entendimento deste projeto, já que o mesmo faz uso dos diagramas de classes e de pacotes.

2.4.1 Ciclo de Vida do Projeto

O ciclo de vida de um projeto descreve as etapas a serem cumpridas desde a concepção de um problema, até sua efetiva implantação num sistema computaci-

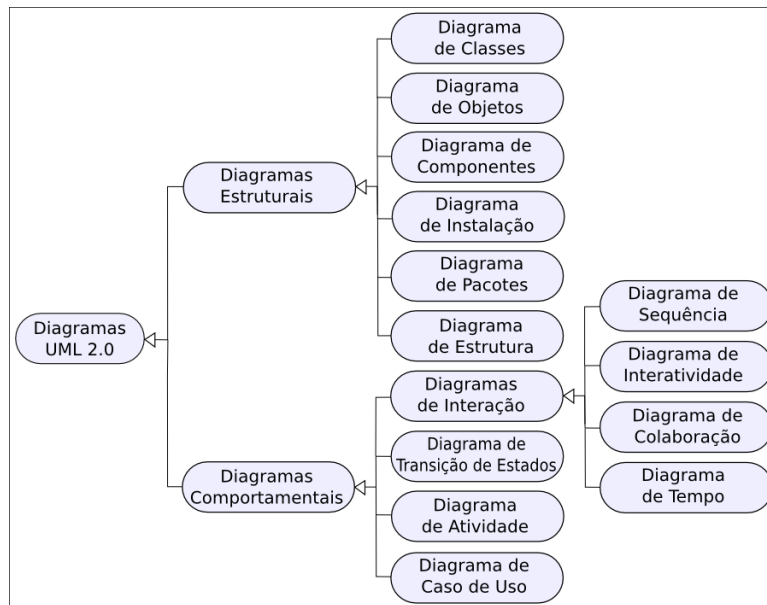


Figura 2.11: Hierarquia dos diagramas UML. Fonte: Adaptado de (BOOCH; JACOBSON; RUMBAUGH, 2006).

onal. De uma maneira genérica podemos citar as seguintes fases de um ciclo de vida:

1. **Levantamento de Requisitos:** Etapa inicial de um projeto de software, onde o desenvolvedor, através de uma entrevista, fará o levantamento de requisitos do software;
2. **Análise:** Através dos requisitos se determinará o que precisa ser feito para atender aos requisitos do software;
3. **Projeto:** Esta etapa se dá após saber o que fazer, que foi definido acima, na Análise. Aqui há um maior detalhamento de como será modularizado o programa;
4. **Implementação:** Esta etapa é onde ocorre a codificação propriamente dita do software, de acordo com o projeto especificado;

5. **Testes:** Esta parte é onde se faz a validação do código em busca de *bugs* no sistema;
6. **Aceitação e implantação:** Após o sistema já estar em uma versão estável (ou quase) e funcional, se instala o programa para que os usuários do sistema possam utilizar o mesmo;
7. **Manutenção:** Esta etapa ocorre após os usuários estarem utilizando o sistema, possíveis *bugs* são corrigidos e novas funcionalidades são implementadas nesta etapa.

3 METODOLOGIA

Esta seção apresenta as características metodológicas do trabalho, como por exemplo: a classificação do tipo de pesquisa; como, onde e a partir de quais ferramentas foi realizado o desenvolvimento deste trabalho.

3.1 Classificação do Trabalho

A metodologia utilizada neste trabalho pode ser classificada de várias maneiras. (JUNG, 2004) propôs uma classificação de tipos de pesquisa científica, conforme a natureza, os objetivos, os procedimentos e o local da pesquisa. A figura 3.1 apresenta um diagrama desta classificação.

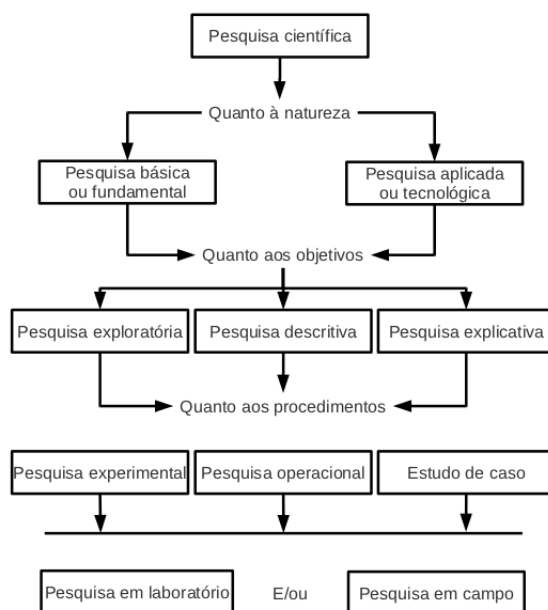


Figura 3.1: Classificação dos tipos de pesquisa. Fonte: (JUNG, 2004).

O presente trabalho apresenta as descrições do tipo de pesquisa, baseada em (JUNG, 2004) e (MARCONI; LAKATOS, 2003). As classificações do trabalho conforme tal modelo são apresentadas a seguir:

1. **Quanto à natureza:** Aplicada ou tecnológica. O presente trabalho lida com a criação de uma nova ferramenta para RSSF.
2. **Quanto aos objetivos:** Exploratória. A proposta é a construção de um software que substitua o antigo programa utilizado e modifique a forma de lidar com visualizações computacionais.
3. **Quanto aos procedimentos:** Experimental. O trabalho visa o desenvolvimento de um programa e formas de organizar melhor a visualização das simulações.
4. **Local da Pesquisa:** Em laboratório. Todo o desenvolvimento do programa foi feito utilizando o computador e softwares de programação.

3.2 Desenvolvimento do Trabalho

A linguagem de desenvolvimento do visualizador foi a linguagem Java. A mesma foi escolhida por possuir as seguintes características. Java é uma linguagem distribuída através da licença GNU *General Public License*, e suporta os principais conceitos de orientação a objetos. Favorece extensibilidade e reusabilidade de código. É altamente portátil, aplicações funcionam do mesmo jeito em qualquer ambiente completamente especificado, ou seja, é independente de plataforma. Suporta aplicações concorrentes fazendo uso de *multithreads*. Possui grande quantidade de recursos disponíveis através de suas APIs - *Application Programming Interface*.

O visualizador proposto foi desenvolvido focando em fazer um programa de simulações que seja fácil de usar, fácil de acrescentar novas funcionalidades. Para conquistar tais objetivos foi utilizado o padrão de projeto MVC (*Model-View-Controller*) (GAMMA *et al.*, 1994). Este padrão de projeto foca na completa separação entre os elementos da interface gráfica, regras de negócio e estrutura de dados.

Durante o desenvolvimento deste trabalho, foram utilizados partes do modelo RUP. Assim fez-se uso dos diagramas UML propostos no RUP, entre estes diagramas se utilizou os diagramas de classes e de pacotes que são partes dos diagramas estruturais. O ciclo de vida do software desenvolvido evoluiu e passou pelas sete fases: levantamento de requisitos, análise, projeto, implementação, testes, aceitação e implantação e por fim manutenção do software.

4 DESENVOLVIMENTO DO TRABALHO

Esta seção tem por objetivo apresentar o desenvolvimento do trabalho como um todo. Assim, a seguir será explicado como foi desenvolvido o *Visual GrubiX* e mais detalhes sobre as decisões de projeto como: qual o modelo de desenvolvimento de software escolhido, o levantamento de requisitos, a estrutura do arquivo de *log*, os diagramas de pacotes e classes, arquitetura MVC, entre outras informações e etapas presentes no desenvolvimento do trabalho.

4.1 Visual GrubiX

Este trabalho, assim como a maioria dos projetos, obedece a todas as sete etapas do ciclo de vida de um projeto descritas na seção 2.4.1. Neste trabalho escrito será dado, atenção especial às etapas de Projeto e de Implementação.

Entre os vários tipos de modelos de desenvolvimento de software existentes como cascata, espiral, evolucionário, incremental, interativo, prototipação etc. Este trabalho optou por utilizar o Modelo Incremental. As etapas necessárias ao desenvolvimento do *Visual GrubiX* estão descritas a seguir:

4.1.1 Levantamento de Requisitos

A ideia básica do visualizador proposto é fazer uma interface gráfica intuitiva para visualização da simulação gerada pelo *framework GrubiX*. Para abrir uma simulação deverá ser carregado um arquivo XML contendo os dados da simulação, em seguida estes dados devem ser analisados e exibidos na tela do visualizador. O visualizador deverá mostrar a área a ser monitorada, os nós da rede e todos os elementos importantes na simulação que foram programados no *GrubiX*. Deverá

possuir controles da simulação e da animação, entre opções de mapeamento dos elementos simulados.

O software deverá conter os seguintes requisitos funcionais coletados. Tais requisitos foram coletados sobre demanda, ou seja, a medida do necessário foram sendo encontrados novos requisitos e incrementados a lista de requisitos e estão listados abaixo.

- Desenvolver uma interface gráfica para o software de visualização de simulações computacionais em RSSF.
- Desenvolver recurso para dar suporte à leitura de arquivos XML no formato gerado pelo *GrubiX*.
- Desenvolver recurso para abrir e fechar uma aplicação gerada pelo *GrubiX*.
- Desenvolver recursos para exibir informações e dados da simulação, como dimensões do ambiente monitorado, quantidade de nós estáticos e móveis, quantidade total de nós, quantidade total e atual do evento simulado, tempo total e atual da aplicação, quantidade total e atual de pacotes enviados, etc.
- Desenvolver recurso para fazer o mapeamento de *cores* e *labels* associados aos nós da rede de acordo com a especificação do arquivo XML da aplicação.
- Desenvolver recurso de *zoom* e translação no painel onde será mostrada a simulação.
- Desenvolver componente para exibir a barra de execução da simulação. Esta barra deve poder ser manipulada de forma dinâmica durante a simulação, alterando assim o fluxo de execução da simulação.
- Desenvolver recurso para visualizar o grafo de comunicação da rede.

- Desenvolver recurso para mostrar a régua de dimensões do mundo simulado.
- Desenvolver recurso para aumentar e diminuir o tamanho dos nós da rede.
- Desenvolver recurso para controle da simulação: iniciar, pausar, recomeçar, voltar a simulação, além de recursos de aumentar e diminuir a velocidade de apresentação.
- Desenvolver nós como componentes selecionáveis, quando um determinado nó é selecionado são exibidas informações como: ID, raio de alcance do sensor, raio de comunicação entre outras.
- Desenvolver recurso que apresente a rota dos nós móveis da rede em linha tracejada.
- Desenvolver recurso para selecionar opção de idiomas no software (português e inglês).
- Desenvolver recurso de atalhos para os principais botões do software.

4.1.2 Estrutura do Arquivo de Log

O formato do arquivo de *log* gerado pelo simulador é o XML que é um padrão muito utilizado no armazenamento de dados (DEITEL *et al.*, 2003). O XML é uma linguagem de marcação para necessidades especiais. Uma linguagem de marcação é um conjunto de códigos aplicados a um texto ou a dados, com o fim de adicionar informações particulares sobre esse texto ou conjunto de dados, ou sobre trechos específicos. Como característica importante deste formato tem-se o fato de ser armazenado em estruturas de árvores. O arquivo gerado pelo *GrubiX* possui o formato esquematizado na Figura 4.1. Este arquivo é carregado pelo visualizador, em seguida é construída uma lista de eventos a serem exibidos na tela do programa.

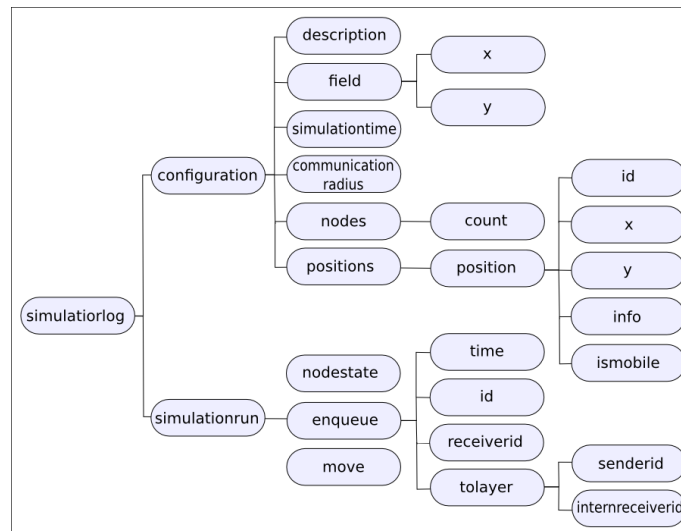


Figura 4.1: Estrutura do arquivos XML gerado pelo simulador.

Conforme visto na Figura 4.1 todas as estruturas estão dentro do campo SIMULATORLOG, como subcampos tem-se a parte de configurações contidos na estrutura CONFIGURATION e tem-se também a simulação propriamente dita no campo SIMULATIONRUN. O campo FIELD é usado para guardar as dimensões do ambiente simulado. O campo COUNT é usado para indicar o número de nós da simulação corrente. Em POSITIONS guardam-se as coordenadas e os IDs de todos os nós da rede. Já na estrutura NODESTATE é feito um mapeamento de todos os nós da rede, para representar a ocorrência de algum evento no ambiente definido pelo programador. O campo ENQUEUE armazena as informações de comunicação, ou seja, como são feitos os envios de pacote na rede. Em MOVE são guardadas as informações de nós móveis e/ou intrusos que se deslocam na rede.

O Anexo A mostra um exemplo básico de como é o arquivo XML utilizado nas leituras do visualizador.

4.1.3 Arquitetura em Camadas

A exibição de dados na tela de simulação é baseada em camadas. Uma arquitetura *multi-layer* é mais apropriada a este tipo de visualização, pois há uma hierarquia bem definida na forma como os mesmos devem ser desenhados na tela. A Figura 4.2 apresenta esta hierarquia.

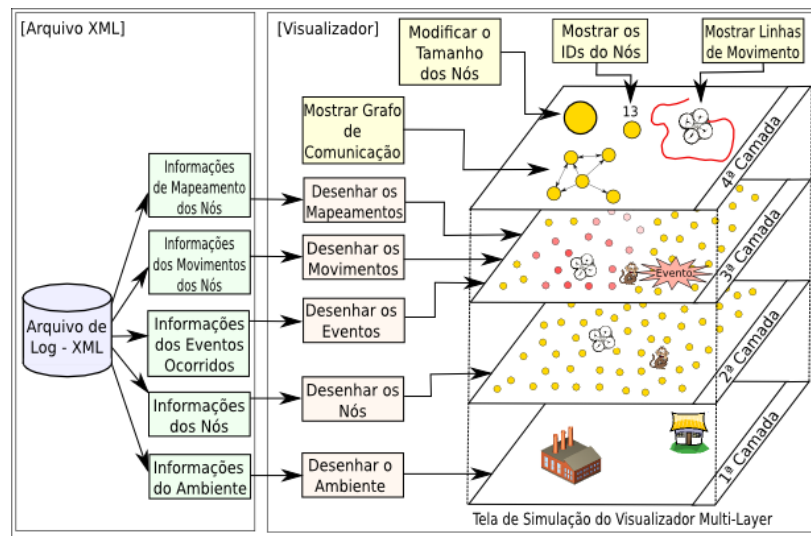


Figura 4.2: Arquitetura em camadas do visualizador.

A primeira camada é a responsável por desenhar na tela o ambiente, de acordo com as informações obtidas do mesmo no arquivo XML. A segunda camada é a responsável por desenhar os nós na tela, isto com base nas informações do arquivo dado. A terceira camada mostra os disparos de eventos, os movimentos dos nós móveis e também faz o mapeamento definido pelo programador e selecionado pelo usuário do programa de visualização. A quarta camada exibe informações dinâmicas na tela, como mostrar o grafo de comunicação, modificar o tamanho dos nós, mostrar a linhas de movimento.

Uma questão a ser notada é que a quarta camada é uma camada independente do arquivo XML. Todas as informações contidas nela não provêm do ar-

quivo XML, sendo assim, são processadas localmente e impressos na tela. Vários recursos podem ser adicionados na quarta camada, pois a mesma tem um alto grau de desacoplamento com as outras camadas.

4.1.4 Diagrama de Pacotes

Os diagramas em geral auxiliam a entender o funcionamento dos sistemas como um todo. Existem vários tipos de diagramas, o primeiro diagrama utilizado aqui é o diagrama de pacotes. Este diagrama auxilia a exemplificar a estrutura lógica do projeto. Um diagrama provê uma parcial representação do sistema. Ele ajuda a compreender a arquitetura do sistema desenvolvido.

O Diagrama de pacotes é um diagrama estático e é definido pela UML, onde descreve os pacotes ou pedaços do sistema divididos em agrupamentos lógicos mostrando as dependências entre estes, ou seja, pacotes podem depender de outros pacotes. Este diagrama é muito utilizado para ilustrar a arquitetura de um sistema mostrando o agrupamento de suas classes. Assim um pacote ajuda a organizar seu projeto, não deixando suas classes espalhadas pelo sistema. Um pacote representa um grupo de classes (ou outros elementos) que se relacionam com outros pacotes através de uma relação de dependência. Um diagrama de pacotes pode ser utilizado em qualquer fase do processo de modelagem e visa organizar os modelos.

A Figura 4.3 mostra o diagrama de pacotes do programa desenvolvido. Neste diagrama são vistos três pacotes principais, que são: CONTROLLER, VIEW e RESOURCES. O pacote CONTROLLER faz o controle da programa em geral e contém a lógica de negócio do mesmo. No pacote VIEW é desenvolvida a interface gráfica e o controle do que é exibido na tela do software. O pacote RESOURCES contém as imagens e arquivos de idiomas utilizadas no programa.

Analisando ainda a Figura 4.3 tem-se no pacote principal de controle (CONTROLLER) os sub-pacotes: CONTROL que contém as principais classes de controle do pro-

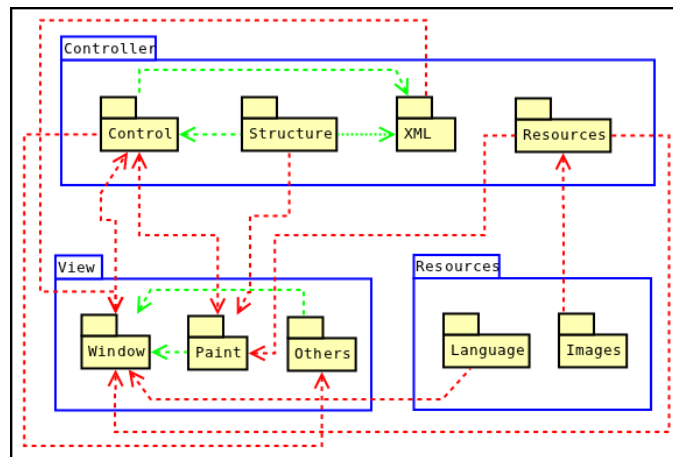


Figura 4.3: Diagrama de pacotes do *Visual GrubiX*.

grama, STRUCTURE possui as classes da estrutura da rede, XML que contém as classes que manipulam os arquivos XML lidos pelo visualizador e RESOURCES que possui uma classe que faz o carregamento dos recursos gráficos do programa.

No pacote de visualização (VIEW) tem-se os sub-pacotes: WINDOW que contém as principais classes de controle das janelas gráficas software, PAINT que possui as classes que pintam a animação da simulação e alguns recursos dentro do painel de visualização da simulação, OTHERS que contém algumas classes que auxiliam na construção da interface gráfica.

O último dos pacotes principais (RESOURCES) não contém classes, mas sim recursos. Este possui como sub-pacotes: LANGUAGE sub-pacote que possui os arquivos de idiomas, já que o visualizador desenvolvido tem acesso aos idiomas iniciais português e inglês; IMAGES que possui os arquivos de imagens utilizadas no programa.

Esta Figura 4.3 mostra também as dependências entre os pacotes, representada pelas setas. As setas verdes indicam dependências entre pacotes dentro de um mesmo pacote-pai. Já as setas vermelhas mostram dependências entre pacotes de

escopos diferentes, ou seja, os pacotes relacionados não possuem o mesmo pacote-pai.

4.1.5 Diagrama de Classes

Conforme dito anteriormente os diagramas servem para auxiliar no entendimento do sistema. E o diagrama de classes é um dos diagramas que melhor auxilia neste objetivo. Sendo assim este diagrama é uma representação da estrutura e relações entre as classes que servem de modelo para objetos. É uma modelagem muito útil para o sistema, define todas as classes que o sistema necessita possuir.

O diagrama de classes é o mais importante diagrama da UML, ele está no centro da sua arquitetura e a partir desse diagrama outros diagramas são elaborados. O diagrama de classes é uma importante ferramenta para a documentação de um sistema ou produto de software, conforme ressaltado por (MELO, 2002).

Segundo (FOWLER; SCOTT, 2000), “Um diagrama de classes descreve os tipos de objetos no sistema e os vários tipos de relacionamentos estáticos que existem entre eles”. As classes representam as propriedades e o comportamento de um conjunto de objetos em um sistema e, conseqüentemente, como essas classes não existem sozinhas, é importante também representar os seus relacionamentos.

Devido a esta série de benefícios gerados por este diagrama, o presente trabalho faz uso dos diagramas de classes para ilustrar melhor o software desenvolvido. A Figura 4.4 mostra o diagrama de classes dos pacotes CONTROL à esquerda e STRUCTURE à direita. Nestes diagramas mostra-se apenas o relacionamento das classes internas ao pacote nos quais estão inseridas, ou seja, CONTROL e STRUCTURE respectivamente. É bom ressaltar que nos diagramas mostrados foram omitidos os atributos, os métodos e a multiplicidade das associações afim de dar maior simplicidade ao diagrama.

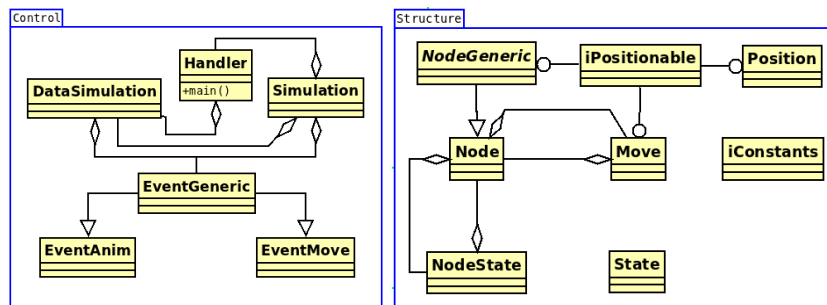


Figura 4.4: Diagramas de classes dos pacotes *Control* e *Structure*.

A Figura 4.5 apresenta o diagrama de classes dos pacotes WINDOW à esquerda e PAINT à direita. Assim como na figura anterior é apresentado apenas o relacionamento das classes internas ao pacotes o qual estão inseridas.

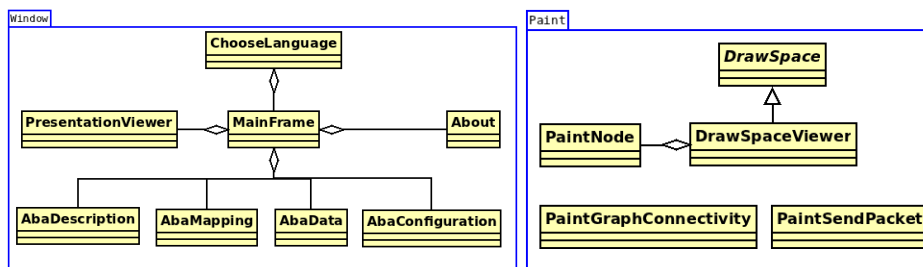


Figura 4.5: Diagramas de classes dos pacotes WINDOW e PAINT.

Estes diagramas apresentados direcionaram todo o desenvolvimento da programação do visualizador. Os diagramas de classes completos mostrando os relacionamentos entre os pacotes pode ser visto no Anexo B.

4.1.6 Arquitetura MVC

O *Visual GrubiX* foi desenvolvido de forma que seja fácil de usar e permita o acréscimo de novas funcionalidades de modo fácil. Para conquistar tais objetivos foi utilizado o padrão de projeto MVC (GAMMA *et al.*, 1994). Este padrão de

projeto prioriza a completa separação entre os elementos da interface gráfica, as regras de negócio e a estrutura de dados.

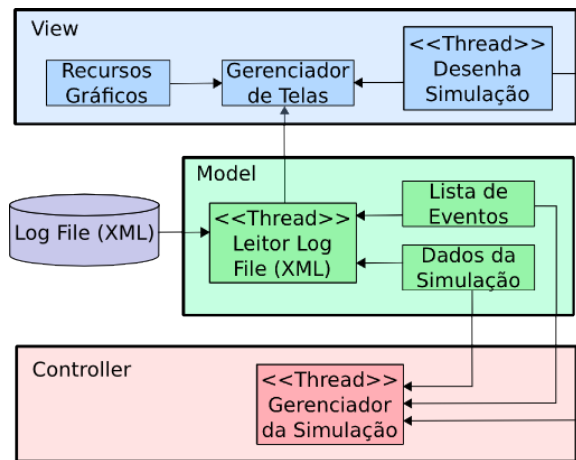


Figura 4.6: Arquitetura MVC do *Visual GrubiX*.

A Figura 4.6 mostra um esquema da arquitetura MVC utilizada no *Visual GrubiX*. Na parte de *Model* tem-se as estruturas que fazem o carregamento do arquivo XML presente no disco e que organizam os dados lidos no arquivo em uma lista de eventos e uma estrutura de dados da simulação. Na parte de *Controller* a estrutura Gerenciador da Simulação cuida de organizar as informações para serem exibidas na tela do visualizador com base nas informações obtidas da lista de eventos e dos dados da simulação obtidas da parte *Model*. Na parte de *View* a estrutura Gerenciador de Telas cuida de mostrar a interface das principais telas do visualizador. Para isto ela faz uso dos recursos gráficos e da estrutura de desenhar na tela a simulação corrente.

4.1.7 Algoritmo que Gerencia a Simulação

Durante a confecção deste visualizador vários algoritmos foram desenvolvidos. A versão atual do programa conta com cerca de 7500 linhas de código. Este possui também cerca de 40 classes e 12 pacotes.

Entre os códigos produzidos alguns fazem controle das estruturas de dados, outros da interface gráfica e do arquivo XML e alguns da simulação em si. Um dos códigos mais importantes é o que faz o controle da simulação dos eventos da rede. Sendo assim um importante trecho deste código está apresentado a seguir.

Algoritmo 1: Algoritmo para simulação dos eventos da rede.

Entrada: ModoDaSimulação: mode, ListaDeEventos: listEvent
Saída: Simulação dos eventos da rede

```

1 início
2   inteiro indexEvent = 0;
3   enquanto (true) faça
4     se (indexEvent <= listEvent.size ()-1 and indexEvent >=0)
5       então
6         EventGeneric event = listEvent.get (indexEvent);
7         se (event instanciade EventMove) então
8           | event.runMove ();
9         fim se
10        se (event instanciade EventMsg) então
11          | event.runMsg ();
12        fim se
13        se (mode = PLAY) então
14          | indexEvent = indexEvent + 1;
15          se (indexEvent > listEvent.size () - 1) então
16            | mode = PAUSE;
17          fim se
18        fim se
19        se (mode = BACK) então
20          | indexEvent = indexEvent - 1;
21          se (indexEvent < 0) então
22            | mode = PAUSE;
23          fim se
24        fim se
25        enquanto (mode = PAUSE) faça
26          | wait ();
27        fim enqto
28      fim enqto
29 fim

```

Observando o Algoritmo 1 vê-se que a simulação pode se encontrar em três modos que são PLAY, BACK ou PAUSE estes modos indicam simulação avançando, retrocedendo ou parada. Quando a simulação está avançando ou retrocedendo, um evento da lista de eventos é capturado e a sua animação é disparada. Este evento pode ser de dois tipos: evento de mensagem ou evento de movimentação dos nós. Após verificar qual é o tipo do evento atual, o mesmo é executado. Caso seja um evento de mensagem a mesma é enviada, caso seja um evento de movimento o nó se move.

4.1.8 Principais Recursos Desenvolvidos

O *Visual GrubiX* contou com todos os recursos designados no levantamento de requisitos. Assim o visualizador é capaz de mostrar toda a simulação de eventos ocorridos na rede. A transmissão de pacotes na rede é feita através de animações, onde cada pacote é entregue ao respectivo nó sensor destino. O suporte aos idiomas português e inglês foi feito usando classes para internacionalização de código java, usando dois arquivos de *properties*. Em cada um destes arquivos tem-se todo o conjunto de palavras em português em um e no outro as respectivas palavras em inglês.

Alguns outros recursos não presentes no levantamento de requisitos foram desenvolvidos como: vários temas, ativar e desativar *anti-aliasing*, mudar forma de envio de pacotes, mostrar os identificadores dos nós da rede, carregar uma imagem de fundo do ambiente, tirar *print-screen* do painel de visualização da simulação etc.

A seguir são encontrados alguns recursos presentes no software de visualização desenvolvido e seu funcionamento.

4.1.8.1 Mapping

O *mapping* ou mapeamento é um recurso presente no *Visual GrubiX*. Este recurso permite ao usuário especificar quais elementos gráficos que representam um determinado aspecto de simulação. Esses aspectos a serem mapeados são definidos pelo programador da rede no momento da codificação do comportamento da rede.

Um exemplo de *mapping* bastante utilizado ocorre quando deseja-se mapear algum(s) nó(s) sensor(es) com informações de algum tipo de pacote especial que foi recebido por algum outro nó. Assim cria-se um *mapping* para reconhecer que este determinado nó recebeu tal pacote. Este *mapping* será apresentado de forma gráfica no visualizador. Esta forma pode ser definida através de cores para os nós, cores das bordas dos nós, ou ainda utilizando *labels* que exibirão informações numéricas.

Conforme dito este mapeamento pode auxiliar o programador a depurar seu código e verificar o perfeito recebimento de pacotes especiais a algum grupo de nós da rede. É importante ressaltar também que o *framework GrubiX* é um dos poucos simuladores que dá suporte a este tipo de recurso (LESSMANN; HEIMFARTH, 2008).

4.1.8.2 Grafo de Conectividade

Uma recurso adicional neste visualizador em relação ao *Visual ShoX* é o modo de exibição onde é mostrado o grafo de conectividade da rede. Este grafo é montado da seguinte forma: linhas direcionais a partir de cada nó para todos os seus vizinhos são exibidas na tela, ou seja, estas ligações indicam o alcance de transmissão de dados dos nós da rede. Este alcance como se sabe se dá de acordo com o alcance do sinal de rádio. Assim caso o grafo da rede seja todo conexo então a rede

estará inteiramente conectada. Na Figura 4.7 vê-se um esquema do funcionamento deste recurso.

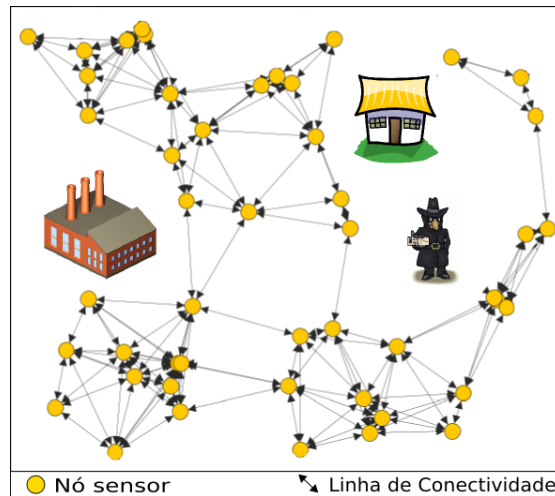


Figura 4.7: Exemplo de grafo de conectividade da rede.

Na Figura 4.7, vê-se que a rede é conexa, já que existe um caminho de um nó para qualquer outro nó da rede. Caso a rede fosse desconexa poderia ser visto que não existiria tal caminho.

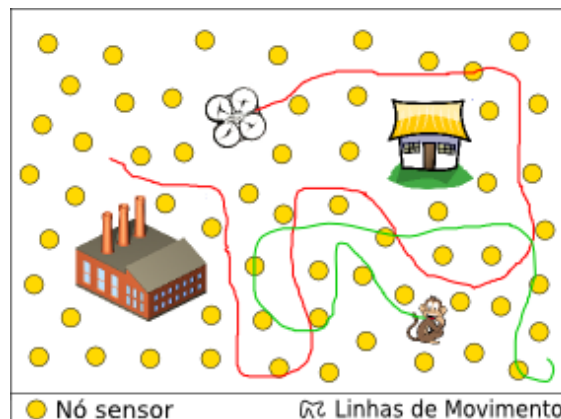


Figura 4.8: Exemplo de linhas de movimentação da rede.

4.1.8.3 Linhas de Movimentação

Um dos principais objetivos deste trabalho é dar uma maior quantidade de informação de uma simulação qualquer ao programador da rede. Sendo assim foi desenvolvido um mecanismo onde são mostradas as linhas de movimentação de nós móveis. O nós móveis de uma rede geralmente são VANTs - Veículos Aéreos Não Tripulados ou ainda intrusos (ou carros, ou pessoas com celular etc). A Figura 4.8 mostra um esquema sobre estas linhas de movimentação que foi desenvolvido para o deslocamento do VANT. Nesta Figura, vê-se em uma das linhas a trajetória indicando onde o VANT passou. Já a outra linha fornece informações da trajetória feita pelo intruso, neste exemplo, um macaco. Essas trajetórias e rotas dão informações ao programador a longo prazo sobre o deslocamento destes objetos. No caso do VANT essa informação pode ser usada para se ter noção global dos algoritmos de deslocamento do mesmo. Já no caso do intruso essa informação pode mostrar na rede uma possível tendência no algoritmo de deslocamento dos intrusos, já que os mesmos também são modelados em software através do *framework*.

5 RESULTADOS E DISCUSSÕES

Nesta seção são apresentados os resultados obtidos e são feitas algumas discussões sobre o software desenvolvido.

Como resultado deste trabalho tem-se o visualizador de simulações computacionais de Rede de Sensores Sem Fio para o *framework GrubiX*, conhecido como *Visual GrubiX*. Um dos principais focos é a criação de uma interface de exibição que permita operação de forma intuitiva. Sendo assim, foi construída uma interface análoga aos softwares de exibição de vídeos no que diz respeito aos botões de controle da simulação. A navegação do software é análoga aos programas de manipulação de imagens e CADs (*Computer-Aided Design*) mais utilizados no mundo, ou seja, para navegar pelo ambiente de simulação basta clicar os botões do mouse, arrastar o mouse e girar o *scroll* do mouse. Esses comandos do mouse permitem fazer *zoom* e transladar o sistema. O programa desenvolvido possui suporte a dois idiomas o português e o inglês e pode ser ampliado com facilidade, uma vez que foram desenvolvidos usando algumas classes de internacionalização do Java.

A Figura 5.1 mostra o produto final deste trabalho. Essa figura mostra mais especificadamente a tela inicial de abertura do software. Nessa tela veem-se as opções para abrir um arquivo de simulação, abrir em tempo real e fechar o programa, além de opções para ir ao menu de ajuda e mudar o tema.

A tela principal do programa desenvolvido pode ser vista na Figura 5.2, onde é apresentada uma simulação de uma RSSF ocorrendo. A interface deste software é composta de quatro áreas. Na área acima há uma barra de menus e uma barra de botões de controle de projetos abertos. Mais a acima e a direita pode-se ver alguns *status* da simulação corrente.

Na área a esquerda vê-se um painel com várias opções de visualização da simulação. Entre essas opções tem-se: habilitar o grafo de comunicação dos nós

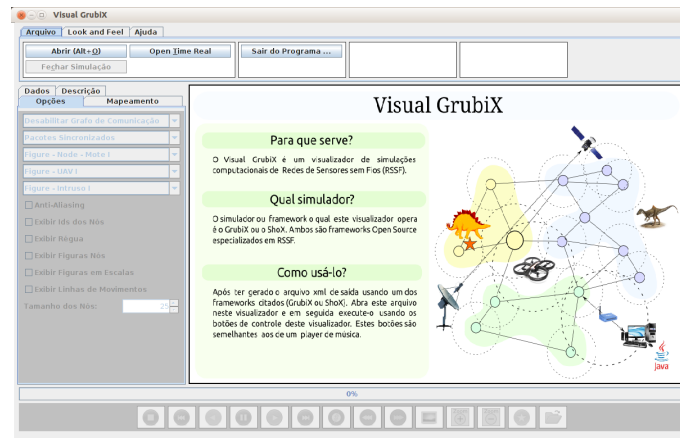


Figura 5.1: Tela inicial do *Visual GrubiX*.

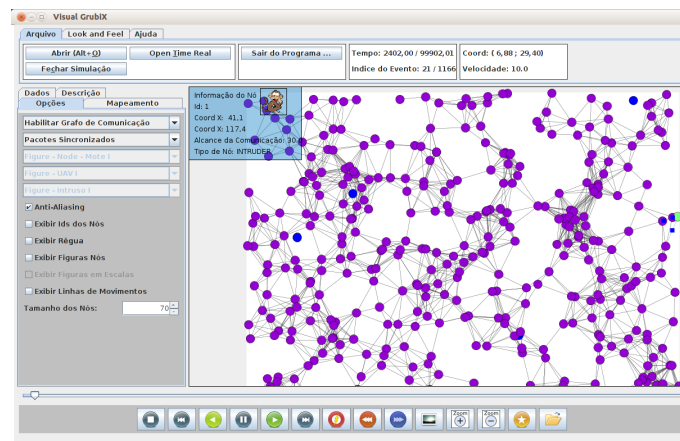


Figura 5.2: Tela de simulação do *Visual GrubiX*.

da rede, mudar a forma de envio de pacotes, selecionar imagens para os nós da rede, habilitar também o ID dos nós da rede, a régua, as linhas de movimentação, entre outros recursos. Pode-se também mudar o tamanho dos nós da rede. Na esquerda tem-se também outras abas uma para fazer o mapeamento dos eventos de interesse ocorridos na rede em uma representação gráfica pretendida, e tem-se a aba de dados que mostra várias informações da rede simulada como densidade

de nós, número médios de nós vizinhos, conectividade da rede, número de pacotes enviados até determinado momento, entre outras informações.

Na área central vê-se uma aplicação sendo executada. Nesta área mostram-se os nós, o grafo de comunicação entre os mesmos, mostra-se um nó selecionado, para o qual ao lado esquerdo e acima são apresentados informações. Nas bordas da área central vê-se uma régua para se ter noção de distâncias e da área de cobertura da rede.

Na área abaixo têm-se os botões de controle da simulação e da animação, onde é possível iniciar, pausar, continuar, e reiniciar o processo de visualização da simulação. Além disso, há vários níveis de velocidade da animação, entre outros recursos. Um pouco acima desses botões tem-se uma barra de progresso da simulação corrente, essa barra é dinâmica onde pode-se retroceder ou avançar e também selecionar um ponto específico de tempo, arrastando o controle deslizante do tempo.

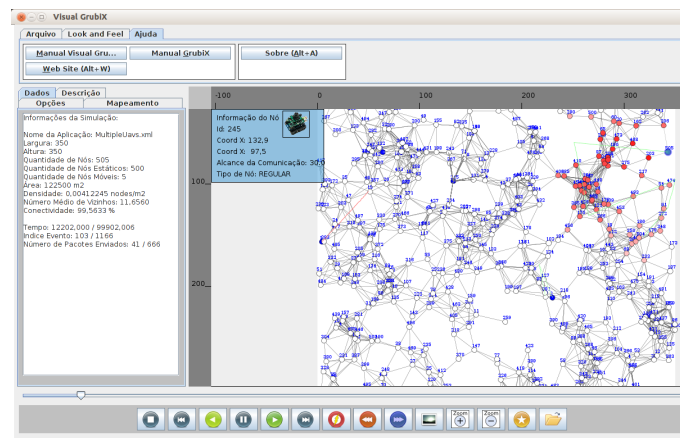


Figura 5.3: Tela de simulação do *Visual GrubiX*.

A Figura 5.3 mostra outra tela onde a mesma simulação da anterior está em curso, com algumas outras configurações selecionadas. Na aba Dados na esquerda veem-se alguns dados da simulação corrente.

Recursos \ Visualizador	<i>Visual GrubiX</i>	<i>Visual ShoX</i>
Multiplataforma	Sim	Sim
Possui mapeamento de elementos	Sim	Sim
Nós da rede selecionáveis	Sim	Não
Mostra grafo conectividade	Sim	Não
Mostra linhas de movimentação	Sim	Não
Mostra imagens nós	Sim	Não
Possui análise estatística	Não	Sim
Definir as configuração da rede	Sim	Não
Rapidez carregar arquivo	Muito rápido	Lento
Botões de controle	Bom	Regular
Controle da animação	± velocidade	± velocidade
Usabilidade do sistema	Fácil	Difícil
Idiomas	Inglês / Português	Inglês
Abrir arquivos <i>log</i>	xml	xml / <i>Compact</i>
Carregamento dos arquivos	Memória / <i>buffer</i>	Memória
<i>ProgressBar</i> simulação	Dinâmica	Estática
Forma de zoom	Usa mouse ou botões	Usa <i>slider</i>
Forma de translação	Usa mouse ou botões	Usa <i>slider</i>

Tabela 5.1: Comparação do *Visual GrubiX* e do *Visual ShoX*.

A Tabela 5.1 mostra algumas comparações de forma resumida entre as funcionalidades, recursos e características do *Visual GrubiX* e do do *Visual ShoX*. Analisando esta comparação entre os visualizadores vê-se que visualizador proposto atende bem aos objetivos do trabalho, já que conseguiu-se obter um visualizador com interface gráfica bem amigável e que apresenta um sistema de navegação bastante intuitivo, além de possuir vários outros recursos não encontrados no *Visual ShoX*. O software desenvolvido ainda não tem todos os recursos que o *Visual ShoX*, como leitura de arquivos *compact*, que são bem mais econômicos em relação ao espaço de armazenamento em disco e também não faz análise estatística.

A Tabela 5.2 mostra algumas características dos visualizadores *Visual GrubiX*, *Visual ShoX*, *nam* e o Visualizador do *OMNeT++*.

Atributos \ Visualizador	<i>Visual GrubiX</i>	<i>Visual ShoX</i>	<i>nam</i>	<i>Visual. OMNeT++</i>
Linguagem	Java	Java	C++	C++
Licença	<i>Open source</i>	<i>Open source</i>	<i>Open source</i>	Licença Acadêmica
Forma de Visualização	<i>offline</i>	<i>offline</i>	<i>offline</i>	<i>online</i>

Tabela 5.2: Comparação geral dos visualizadores *Visual GrubiX*, *Visual ShoX*, *nam* e *OMNeT++*.

Além dos recursos descritos nos requisitos do software foram, desenvolvidos outras funcionalidades. Entre as novas funcionalidades destacam-se: recurso para ativar ou desativar o *anti-aliasing* (técnica antisserrilhamento embaça as bordas, dando a impressão visual de um contorno mais suave); recurso para mudar a forma de envio de pacotes da rede; recurso para mostrar o identificador (ID) de todos os nós da rede; recurso para mostrar a imagem de todos os nós da rede; recurso para carregar uma imagem de fundo da rede; recurso para tirar *print screen* do painel onde é visualizada a simulação; recurso para exibir vários temas (*look and feel*) da interface e personalizar a aparência do software.

6 CONCLUSÃO E TRABALHOS FUTUROS

Esse trabalho apresentou o *Visual GrubiX*, um visualizador de redes de sensores sem fio, cujo principal foco é permitir visualizar de maneira fácil e intuitiva as aplicações desenvolvidas no *GrubiX* e no *ShoX* (*frameworks* de RSSF). Seus objetivos consistem em manter a compatibilidade com os dois *frameworks*. O fato do *Visual ShoX* e *Visual GrubiX* terem uma certa familiaridade faz com que seja fácil a migração de um visualizador para outro, facilitando o aprendizado. O novo visualizador possui uma interface gráfica amigável.

O *Visual Grubix* apresenta-se como uma versão de visualização *offline* para o *framework GrubiX* escrito em Java. O mesmo possui recursos para mostrar as formas de visualização dos dados como as linhas de movimentação que indicam onde os nós se moveram, grafo de conectividade que exhibe os alcances dos nós sensores. Possui também muitos recursos para controle das simulações como avançar ou retroceder uma simulação, parar, aumentar e diminuir a velocidade, entre outros.

Neste trabalho um dos focos foi desenvolver um software com alta manutibilidade e legibilidade de código, usando os conceitos presentes na orientação a objetos. Esta preocupação com manutibilidade e legibilidade ocorre pelo fato de que pretende-se que novos desenvolvedores possam vir a contribuir implementando novas funcionalidades no *Visual GrubiX*.

O *Visual GrubiX* supera várias deficiências do *Visual ShoX* ao qual faltam vários recursos de visualização. O visualizador proposto utiliza um arquivo XML como entrada para a visualização e permite um visual rico e depuração de simulações de rede. Além disso, o esquema de *mapping* permite ao usuário especificar quais elementos gráficos representam um determinado aspecto de simulação.

Como proposta de trabalhos a serem realizados no futuro podem-se citar: acrescentar novas funcionalidades ao visualizador como aumentar a fidelidade das

simulações integrando o visualizador com o *Google Maps*. Criar um recurso para produzir vídeos da simulação corrente. Aumentar a abrangência do software com suporte a novos formatos de arquivo de saída *.compact*, este formato economiza espaço em disco e é suportado pelo *Visual ShoX*. Por fim, um outro recurso interessante de se implementar é um componente que visualiza a parte do *script XML* que está sendo executada, em um determinado instante de tempo. Este recurso é interessante para que o desenvolvedor da aplicação possa analisar erros.

REFERÊNCIAS

- BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. *UML - Guia Do Usuário*. 2. ed. [S.l.]: Elsevier, 2006. 474 p.
- BUSCHMANN, C.; PFISTERER, D.; FISCHER, S.; FEKETE, S. P.; KROLLER, A. Spyglass: a wireless sensor network visualizer. *SIGBED Rev.*, ACM, New York, NY, USA, v. 2, n. 1, p. 1–6, jan. 2005. ISSN 1551-3688. Disponível em: <<http://doi.acm.org/10.1145/1121782.1121784>>.
- CROSSBOW. *MoteView Users Manual*. Crossbow Technology, 2007.
- DEITEL, H.; DEITEL, P.; NIETO, R.; LIN, T.; SADHU, P. *Livro XML Como Programar*. [S.l.]: Bookman, 2003.
- FOWLER, M.; SCOTT, K. *UML essencial: um breve guia para a linguagem padrão de modelagem de objetos*. 2. ed. [S.l.]: Porto Alegre - RS, 2000. 169 p.
- GAMMA, E.; JOHNSON, R.; HELM, R.; VLISSIDES, J. M.; BOOCH, G. *Design Patterns: Elements of Reusable Object-Oriented Software*. [S.l.]: Addison-Wesley, 1994.
- GRUBIX. *Simulador GrubiX*. 2013. Disponível em: <<http://pesquisa.dcc.ufla.br/grub/grubix-simulator/>>.
- JANACIK, P.; LESSMANN, J.; HEIMFARTH, T.; KARCH, M. Towards an efficient protocol development process in the shox network simulator. In: *Computer Modelling and Simulation (UKSim), 2010 12th International Conference on*. [S.l.: s.n.], 2010. p. 233 –238.
- JUNG, C. F. *Metodologia para pesquisa & desenvolvimento: aplicada a novas tecnologias, produtos e processos*. [S.l.]: Rio de Janeiro - RJ, 2004.

LESSMANN, J.; HEIMFARTH, T. Flexible offline-visualization for mobile wireless networks. In: *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*. Washington, DC, USA: IEEE Computer Society, 2008. (UKSIM '08), p. 404–409. ISBN 978-0-7695-3114-4. Disponível em: <<http://dx.doi.org/10.1109/UKSIM.2008.33>>.

LESSMANN, J.; HEIMFARTH, T.; JANACIK, P. Shox: An easy to use simulation platform for wireless networks. In: *UKSIM '08: Proceedings of the Tenth International Conference on Computer Modeling and Simulation*. Washington, DC, USA: IEEE Computer Society, 2008. p. 410–415. ISBN 978-0-7695-3114-4.

LEVIS, P.; LEE, N.; WELSH, M.; CULLER, D. Tossim: accurate and scalable simulation of entire tinyos applications. In: *Proceedings of the 1st international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2003. (SenSys '03), p. 126–137. ISBN 1-58113-707-9. Disponível em: <<http://doi.acm.org/10.1145/958491.958506>>.

LOUREIRO, A. A. F.; NOGUEIRA, J. M. S.; RUIZ, L. B.; MINI, R. A. de F.; NAKAMURA, E. F.; FIGUEIREDO, C. M. S. Redes de sensores sem fio. In: *Tutoriais do Simpósio Brasileiro de Redes de Computadores*. [S.l.: s.n.], 2003.

MARCONI, M. de A.; LAKATOS, E. M. *Fundamentos de metodologia científica*. [S.l.]: São Paulo - SP, 2003.

MELLO, B. A. d.; CAIMI, L. L. Simulação na validação de sistemas computacionais para a agricultura de precisão. *Revista Brasileira de Engenharia Agrícola e Ambiental*, volume 12, p. 666–675, 2008.

MELO, A. C. *Desenvolvendo Aplicações com UML: do Conceitual à Implementação*. [S.l.]: Rio de Janeiro, 2002. 255 p.

PARBAT, B.; DWIVEDI, A.; VYAS, O. Data visualization tools for wsns: A glimpse. *International Journal of Computer Applications*, Volume 2, 2010.

SHOX. *ShoX - A scalable ad hoc network simulator*. 2012. Disponível em: <<http://shox.sourceforge.net>>.

SILVA, R. C. *Redes de Sensores Sem Fio*. Dissertação (Monografia de Graduação) — Universidade Estadual de Montes Claros, 2006.

TRANSCODE. *Transcode Wiki*. 2012. Disponível em: <<http://www.transcoding.org/cgi-bin/transcode>>.

VARGA, A. *Omnet++ Discrete Event Simulation System*. 2012. Disponível em: <<http://www.omnetpp.org>>.

A ANEXO

No *script* A.1 encontra-se um arquivo XML. Este arquivo mostra um exemplo genérico das informações contidas na saída do *framework GrubiX*. A estrutura deste arquivo foi utilizada como informação para o desenvolvimento do software proposto nesta monografia, o *Visual GrubiX*.

```

1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!--GrubiX XML - Criado por Jesimar-->
3 <simulatorlog>
4   <configuration>
5     <description write="Texto descrevendo a aplicacao." />
6     <field>
7       <x>100.0</x>
8       <y>100.0</y>
9     </field>
10    <simulationtime stepspersecond="100" base="steps">60000</
      simulationtime>
11    <communicationradius>25.0</communicationradius>
12    <nodes>
13      <count>50</count>
14    </nodes>
15    <positions>
16      <position>
17        <id>1</id>
18        <x>34.91</x>
19        <y>32.52</y>
20        <info nodetype="REGULAR" />
21        <ismobile>>false</ismobile>
22      </position>
23    </positions>
24  </configuration>
25  <simulationrun>
26    <nodestate id="3" name="Valor" type="int" value="244" />
27    <move id="5" x="12.5" y="74.3" time="37.2" />
28    <enqueue>
29      <time>37.2</time>
30      <id>561</id>

```

```
31     <receiverid>-1</receiverid>
32     <tolayer>
33         <senderid>1</senderid>
34         <internreceiverid>7</internreceiverid>
35     </tolayer>
36 </enqueue>
37 </simulationrun>
38 </simulatorlog>
```

Script A.1: Exemplo do formato do arquivo XML.

B ANEXO

Abaixo encontram-se alguns diagramas de classes mais completos do visualizador desenvolvido.

A Figura B.1 mostra o diagrama de classes referente ao pacote `CONTROLLER`. Este pacote possui quatro sub-pacotes que são: `CONTROL`, `STRUCTURE`, `XML` e `RESOURCES`. Dentro do pacote `CONTROL` tem alguns classes que fazem o controle do software como um todo. No pacote `STRUCTURE` existem as classes responsáveis por guardar os dados lidos do arquivo XML. O pacote `XML` é o responsável por fazer a leitura do arquivo XML, a qual pode ser feita usando a API do Java DOM ou SAX. O pacote `RESOURCES` contém as classes responsáveis por fazer o carregamento dos recursos gráficos como imagens e ícones.

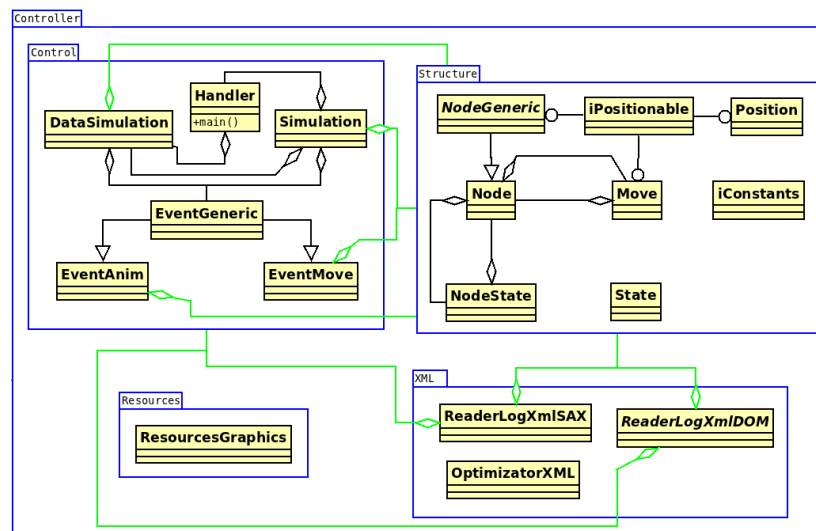


Figura B.1: Diagrama de classes da camada *Controller*.

A Figura B.2 mostra o diagrama de classes referente ao pacote `VIEW`. Este pacote possui também sub-pacotes que são: `WINDOW`, `PAINT` e `OTHERS`. Dentro do pacote `WINDOW`, de maneira geral, existem as classes que são janelas gráfi-

cas ou ainda abas da janela principal. No pacote PAINT existem as classes para desenhar na tela dentro da área da simulação. O pacote OTHERS contém classes genéricas para auxiliar no desenvolvimento da interface gráfica.

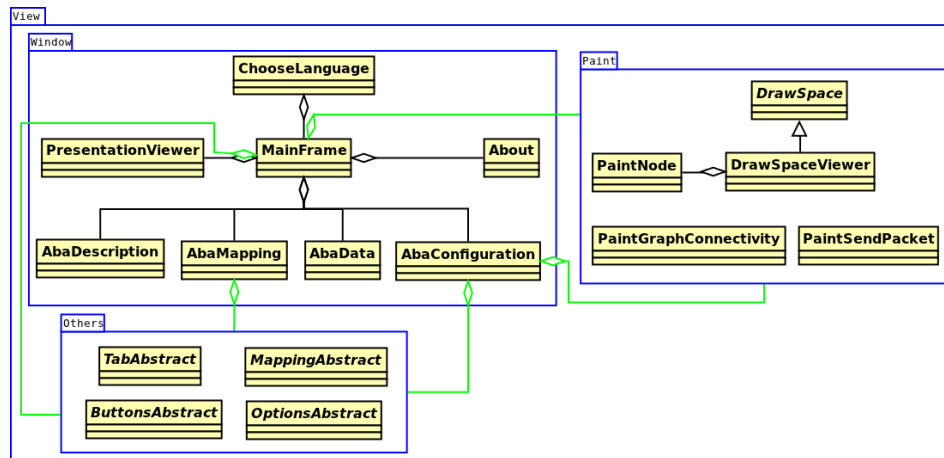


Figura B.2: Diagrama de classes da camada *View*.

A Figura B.3 mostra o diagrama de classes referente a todo o projeto do *Visual GrubiX*. Esta figura, como pode-se observar, é a junção das duas anteriores e assim acaba por mostrar como as mesmas se relacionam.

Estes diagramas podem ser melhor entendidos observando-se as cores dos relacionamentos. As linhas pretas mostram os relacionamentos dentro do mesmo pacote, ou seja, relacionamentos locais. Os relacionamentos com linhas verdes são relacionamentos de classes de pacotes diferentes, mas presos à estrutura de CONTROLLER ou VIEW. Afim de maior simplicidade mostrou-se somente de qual classe sai este tipo de relacionamento, não mostrando a classe destino, pois ficava muito confuso de se entender. Já os relacionamentos com cores vermelhas são entre estruturas diferentes, ou seja, vão do pacote CONTROLLER para o pacote VIEW e vice-versa. São relacionamentos globais onde o controlador comunica-se com a interface gráfica e vice-versa.

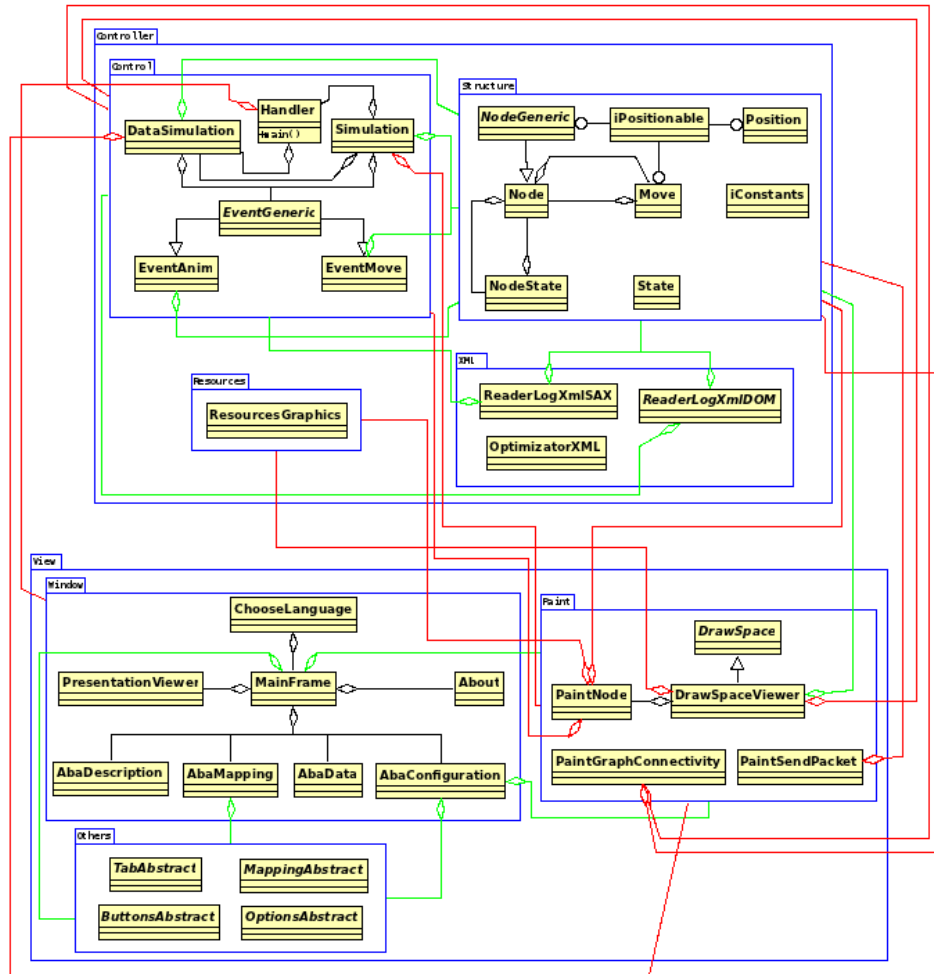


Figura B.3: Diagrama de classes completo do *Visual GrubiX*.