UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ - UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E
DESENVOLVIMENTO DE SISTEMAS

MARCOS DE MELO SIEGA

# INTEGRAÇÃO DAS FERRAMENTAS VANETMOBISIM E NS-3 PARA SIMULAÇÃO DE REDES VANETS

TRABALHO DE DIPLOMAÇÃO

**MEDIANEIRA**

**2013**

MARCOS DE MELO SIEGA

# INTEGRAÇÃO DAS FERRAMENTAS VANETMOBISIM E NS-3 PARA SIMULAÇÃO DE REDES VANETS

Trabalho de Diplomação apresentado ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná - UTFPR - como requisito parcial para obtenção do título de tecnólogo.

Orientador: Prof Dr. Neylor Michel

Co-orientador: Prof Dr. Hermes I. Del Monego

**MEDIANEIRA**

**2013**

Ministério da Educação
**Universidade Tecnológica Federal do Paraná**
Diretoria de Graduação e Educação Profissional
Coordenação do Curso Superior de Tecnologia em
Análise e Desenvolvimento de Sistemas

## TERMO DE APROVAÇÃO

## INTEGRAÇÃO DAS FERRAMENTAS VANETMOBISIM E NS-3 PARA SIMULAÇÃO DE REDES VANETS

Por

**Marcos de Melo Siega**

Este Trabalho de Diplomação (TD) foi apresentado às 9:30 h do dia 23 de agosto de 2013 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. O candidato foi argüido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Dr. Neylor Michel
UTFPR – *Campus* Medianeira
(Orientador)

Prof. Msc. Paulo Lopes de Menezes
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Dr. Jean Metz
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Juliano Rodrigo Lamb
UTFPR – *Campus* Medianeira
(Responsável pelas atividades de TCC)

A folha de aprovação assinada encontra-se na Coordenação do Curso.

À minha família, pelo incansável apoio.

## AGRADECIMENTOS

A Jesus Cristo, por ter me dado saúde e força.

Ao professor, doutor e orientador Neylor Michel, pelo apoio e orientação na elaboração deste trabalho.

Ao professor, doutor e co-orientador Hermes I. Del Monego, pelo apoio e orientação na elaboração deste trabalho.

À minha família, pela confiança e apoio durante toda esta jornada.

Aos professores do curso de Tecnologia em Análise e Desenvolvimento de Sistemas, por todos os ensinamentos passados.

A todos os amigos e colegas que de alguma maneira contribuíram para o desenvolvimento deste trabalho.

“Utilize os talentos que tiver: haveria silêncio nos bosques se só cantassem os pássaros que cantam bem.”

# RESUMO

SIEGA, Marcos. INTEGRAÇÃO DAS FERRAMENTAS VANETMOBISIM E NS-3 PARA SIMULAÇÃO DE REDES VANETS. 44 f. Trabalho de Diplomação – Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná - UTFPR. Medianeira, 2013.

Este trabalho tem como objetivo realizar um estudo acerca da integração da ferramentas VanetMobiSim e NS-3, a fim de simular adequadamente cenários de redes VANETs. O estudo realizado, baseou-se na necessidade de alternativas que auxiliem a experimentação destes cenários, pois os custos envolvidos na implantação destas redes são significativos e o riscos associados ao mal planejamento são enormes.

**Palavras-chave:** Integração de sistemas, Simulação de rede, Mobilidade veicular

## ABSTRACT

SIEGA, Marcos. INTEGRATION OF THE TOOLS VANETMOBISIM AND NS-3 TO SIMULATE VANET NETWORKS SCENARIOS. 44 f. Trabalho de Diplomação – Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná - UTFPR. Medianeira, 2013.

This paper aims to conduct a study on the integration of the tools VanetMobiSim and NS-3 in order to properly simulate VANET networks scenarios. The study was based on the need for alternatives to aid experimentation of these scenarios, because the costs involved in the deployment of these networks are significant and the risks associated with poor planning are enormous.

**Keywords:** System Integration, Network Simulation, Vehicular mobility.

## LISTA DE FIGURAS

## LISTA DE SIGLAS

| | |
|---|---|
| API | Application Programming Interface |
| CSM | Constant Speed Motion |
| FTM | Fluid Traffic Model |
| GBMM | Graph-Based Mobility Model |
| GDF | Geographic Data Files |
| IDM | Intelligent Driver Model |
| IDM_IM | Intelligent Driver Model with Intersection Management |
| IDM_LC | Intelligent Driver Model with Lane Changing |
| ITS | Intelligent Transportation System |
| NS-3 | Network Simulator 3 |
| SMM | Smooth Motion Model |
| TIGER | Topologically Integrated Geographic Encoding and Referencing |
| UML | Unified Modeling Language |
| XML | eXtensible Markup Language |

**SUMÁRIO**

# 1 INTRODUÇÃO

Diversos avanços tecnológicos têm sido agregados aos veículos automotores, que visam melhorar a experiência do condutor e dos passageiros. Alguns exemplos destas tecnologias são os sistemas de frenagem, os sensores capazes de detectar e advertir o condutor da proximidade de objetos e indicadores de velocidade. A maioria destes sistemas são fundamentados em sensores e atuadores que são cada vez mais sofisticados, que fazem com que o veículo seja capaz de captar os sinais e alterações de ambiente e repassar tais informações rapidamente ao condutor. Entretanto, estes sistemas limitam-se à interação entre o veículo e o condutor (ALVES et al., 2009).

O próximo passo da evolução tecnológica consiste em sistemas de comunicação que possibilitem a interação entre diferentes veículos. O objetivo principal desses sistemas é possibilitar a comunicação de usuários móveis e oferecer as condições necessárias para que aplicações com diferentes requisitos sejam atendidas satisfatoriamente. Tais aplicações compõem um Sistema Inteligente de Transporte (*Intelligent Transportation System* - ITS) que opera em um ambiente formado por usuários no trânsito. Exemplos dessas aplicações incluem o monitoramento cooperativo do tráfego e o auxílio a cruzamentos sem sinalização ou a prevenção de colisões. Além das aplicações específicas de trânsito, vislumbra-se o acesso à Internet em qualquer lugar e a qualquer instante (LI; WANG, 2007).

A atuação dos sistemas sobre a comunicação entre veículos formam as chamadas redes veiculares. Estas redes são formadas basicamente pela comunicação entre veículos automotores ou entre veículos automotores e a infraestrutura fixa presente às margens das vias de trânsito. Uma das principais características que a torna diferente das redes sem fio convencionais é a natureza dos nós, que são compostos por automóveis, caminhões, ônibus, etc., com interfaces de comunicação *wireless*, e por estruturas fixas próximas às vias (ALVES et al., 2009).

Nas chamadas Redes Ad hoc Veiculares (VANETs), cada automóvel é um nó ou um roteador. As VANETs são um tipo especial das Redes Ad hoc Móveis (MANETs), em que automóveis equipados com capacidade de processamento e de comunicação sem fio criam uma rede espontânea ao se aproximarem de outros veículos com a mesma característica (ALVES et al., 2009).

Um aspecto crítico presente no estudo de redes VANETs é a necessidade de reproduzir adequadamente o comportamento do cenário móvel antes de realizar efetivamente sua

implantação (LOPES, 2011).

Diante de tal necessidade, este trabalho propõe a realização de um estudo acerca da integração das ferramentas VanetMobiSim e NS-3, utilizando o comportamento de mobilidade simulado no *software* VanetMobiSim em cenários construídos na aplicação NS-3, com o objetivo de simular adequadamente cenários de redes VANETs.

## 1.1 OBJETIVO GERAL

Realizar a integração das ferramentas VanetMobiSim e NS-3 para simulação adequada de cenários de redes VANETs.

## 1.2 OBJETIVOS ESPECÍFICOS

- Realizar estudo acerca do funcionamento da ferramenta VanetMobiSim;
- Realizar estudo acerca do funcionamento da ferramenta NS-3;
- Estudar as formas de integração entre as ferramentas; e
- Validar a integração por meio da criação e exportação de um cenário da ferramenta VanetMobiSim, e importação do cenário no software NS-3.

## 1.3 JUSTIFICATIVA

Encontra-se uma série de desafios para a larga adoção das redes VANETs. Dentre eles destacam-se a alta mobilidade dos nós, a constante mudança dos cenários, a escalabilidade quanto ao número de nós e a garantia de conexão durante a transmissão dos dados, diante do curto tempo em que os nós permanecem em contato. Outro grande desafio encontrado é a realização de experimentos reais, pois resultam em alto custo e necessidade de uma alta demanda de pessoas, por isso a necessidade de simulação (ALVES et al., 2009).

Propõe-se utilizar a integração das ferramentas VanetMobiSim e NS-3 para simular cenários de redes VANETs, que podem auxiliar significativamente na redução do custo com infraestrutura e demanda de pessoas, além de contar com os recursos das ferramentas para compreender a estrutura da rede e dispor deste conhecimento para facilitar a gestão.

## 1.4 ESTRUTURA DO TRABALHO

O trabalho está dividido em cinco capítulos. Logo após a parte introdutória, apresenta-se no segundo capítulo o referencial teórico das tecnologias utilizadas – VANETs, VanetMobiSim, NS-3.

No terceiro capítulo é apresentada a estrutura do estudo experimental desenvolvido utilizando as tecnologias citadas.

No quarto capítulo estão contidos os resultados do desenvolvimento do estudo experimental proposto. As conclusões finais do projeto e sugestão de trabalhos futuros são abordados no quinto capítulo.

# 2 REFERENCIAL TEÓRICO

Neste capítulo é apresentado o referencial teórico correspondente às tecnologias e conceitos utilizados no decorrer do trabalho.

## 2.1 VANETS

*Vehicular Ad hoc Networks* (VANETs) são vistas como uma tipo de *Mobile Ad hoc Networks* (MANETs). Tais redes são distribuídas, contam com comunicações auto-organizadas e são caracterizadas pela mobilidade dos nós que as compõem (CARLINK::UMA, 2007).

Redes VANETs têm algumas caracteristicas especiais, sendo que as principais são relacionadas aos nós, que possuem alta mobilidade e limitado grau de liberdade em relação a mobilidade de nós presentes em redes MANETs. Esta limitação está associada ao fato de que os nós contidos em VANETs devem se mover de acordo com o comportamento real de tráfego de veículos (CARLINK::UMA, 2007).

A natureza deste tipo de rede não permite a utilização de protocolos e aplicações utilizadas em outros tipos de redes. Porém, uma grande quantidade de novas aplicações e protocolos estão surgindo para suprir esta necessidade. A avaliação e teste destes novos recursos estão sendo utilizados em ferramentas de simulação de redes (CARLINK::UMA, 2006).

Segundo CARLINK::UMA (2007), os modelos de mobilidade veicular são geralmente classificados em duas grandes categorias: os modelos de macro-mobilidade e micro-mobilidade.

Essas duas categorias são explicadas nas subseções a seguir.

### 2.1.1 MACRO-MOBILIDADE

Macro-mobilidade descreve todos os aspectos macroscópicos que definem o tráfego veicular: topologia de ruas, restrições aplicadas ao movimento dos veículos, a caracterização das ruas, definindo: limites de velocidade, número de faixas, ultrapassagens e regras de segurança ao longo de cada rua da referida topologia, ou descrição de sinais de trânsito que estabelecem as regras dos cruzamentos (CARLINK::UMA, 2007).

Segundo CARLINK::UMA (2007), as caracteristicas de macro-mobilidade que podem ser definidas para a simulação do comportamento dos veículos são:

- Topologia das ruas: a macro-locomoção é restrita ao movimento em um grafo;
- Os pontos iniciais e de destino;
- A locomoção entre os diferentes pontos de uma estrada;
- A seleção de caminho entre os pontos; e
- A velocidade dos veículos através de diferentes caminhos.

#### 2.1.1.1 TOPOLOGIAS DE ESTRADAS

A seleção deste recurso é um importante fator para que se consiga uma simulação de movimentos veiculares realísticos. Esta característica afeta diretamente algumas medidas importantes, como a velocidade permitida ou a densidade do tráfego por meio de uma estrada (CARLINK::UMA, 2007).

Segundo Härri et al. (2007), as topologias de estradas são definidas por grafos categorizados da seguinte maneira:

- *User-defined*: A topologia é definida como uma lista de vértices interconectados por linhas, Figura 1;
- *Random*: Esta topologia é gerada aleatoriamente utilizando um Diagrama de Voronoi, conforme Figura 2. Segundo Campos e Freitas (1998) , tal diagrama pode ser definido da seguinte maneira: dado um conjunto *S* de *n* pontos no plano, deseja-se determinar para cada ponto *p* de *S* qual é a região *V(p)* dos pontos do plano que estão mais próximos de *p* do que de qualquer outro ponto em *S*. As regiões determinadas por cada ponto formam uma partição do plano chamada de Diagrama de Voronoi;
- *Extracted from real maps*: As informaçãos deste tipo de topologia são obtidas de mapas reais e representadas utilizando diferentes padrões, tais como o *Geographic Data Files* (GDF[1]), que é um padrão utilizado para descrever e transferir dados relacionados a redes rodoviárias e suas vias, ou *Topologically Integrated Geographic Encoding and Referencing* (TIGER[2]), padrão utilizado pelo *United States Census Bureau* para descrever os atributos territoriais (estradas, edifícios, rios e lagos). Sua visualização está na Figura 3.

[1] http://www.ertico.com/gdf-geographic-data-files/#GDF_Introduction
[2] http://www.census.gov/geo/www/tiger/

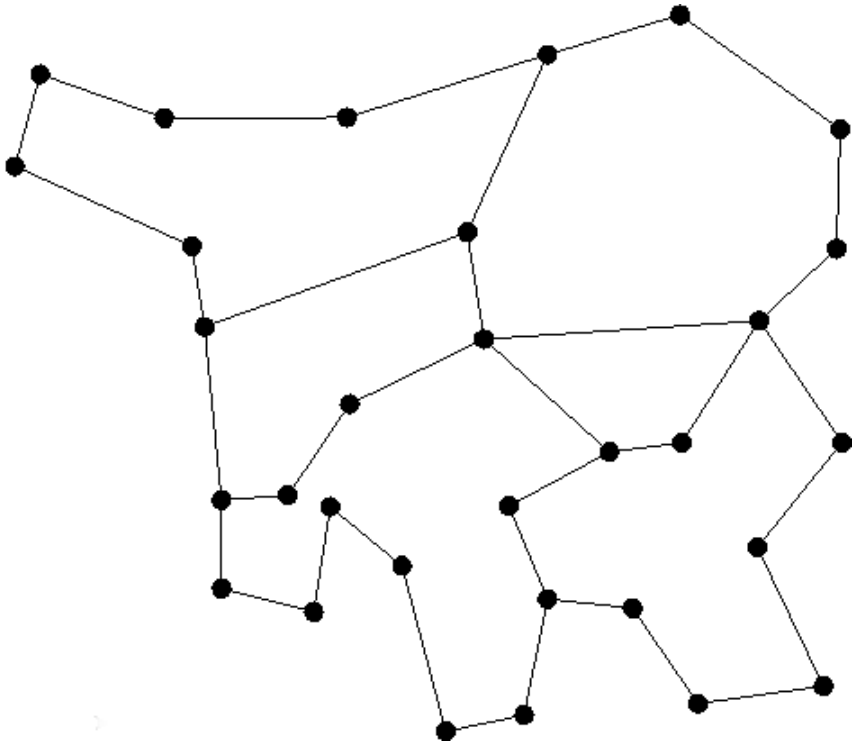


**Figura 1: Topologia User-defined.**

**Fonte: CARLINK::UMA (2007)**

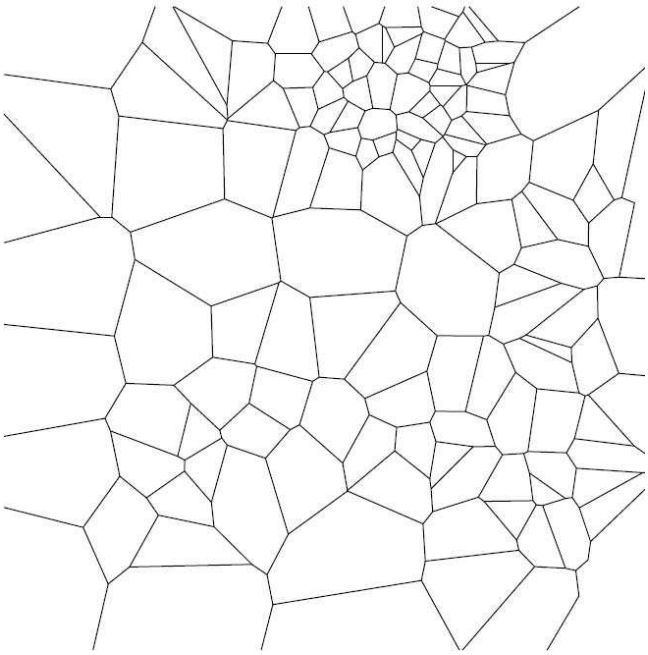


**Figura 2: Topologia Random.**

**Fonte: CARLINK::UMA (2007)**

**Figura 3: Topologia de mapa GDF.**
**Fonte: CARLINK::UMA (2007)**

Segundo CARLINK::UMA (2007), os pontos de partida e destino podem ser definidos utilizando um conjunto de pontos de atração e repulsão, ou seja, quando atribuído ao ponto a característica magnética positiva, este passará a repelir pontos de mesmo atributo e atrair aqueles de qualidade diferente . Além disso, estes pontos podem ser definidos de maneira aleatória ou aleatória restrita.

As viagens são diferentes sequência de pontos pelos quais os veículos passarão durante a simulação. Estes pontos podem ser definidos por sequências de atividades, que são um conjunto de pontos de interesse definidos, por exemplo, através do definição do cenário simulado, ou aleatoriamente.

De acordo com CARLINK::UMA (2007), independentemente do método de geração de viagem utilizado, algoritmos diferentes (por exemplo, algoritmo de *Dijkstra* (FAN; SHI, 2010) e algoritmo de *Bellman-Ford* (AL-KHWILDI; AL-RAWESHIDY, 2006)) podem ser utilizados para gerar a seqüência de arestas visitadas para chegar ao próximo ponto de destino selecionado. Para isso, diferentes aspectos podem ser considerados, por exemplo, o caminho mais curto, o congestionamento do tráfego, etc.

A velocidade dos veículos durante a simulação pode ser limitada pelas características da estrada; estas velocidades podem ser uniformes em todas as estradas, acelerações suaves ou dependentes da via.

### 2.1.2 MICRO-MOBILIDADE

Segundo CARLINK::UMA (2007), o conceito de modelo de micro-mobilidade inclui todos os aspectos relacionados ao comportamento individual do veículos, por exemplo, velocidade e aceleração. Este comportamento está ligado a aspectos pessoais do motorista, como o sexo, a idade ou até mesmo o humor.

A descrição de micro-mobilidade desempenha um papel importante na definição de modelos veiculares realísticos, principalmente porque esses modelos influenciam na variação suave de velocidade, filas de carros, os engarrafamentos e ultrapassagens (FIORE et al., 2007).

De acordo com CARLINK::UMA (2007), os principais aspectos que podem ser representados utilizando recursos de micro-mobilidade são:

- *Human Mobility Patterns*: Os movimentos internos dos carros e suas interações com outros veículos podem ser inspirados em movimentos humanos descritos por modelos matemáticos iguais ao modelo *Car Following*[3].
- *Lane Changing*: Descreve os modos de ultrapassagens implementados pelo modelo, se houver. Um dos modelos que gerenciam as trocas de pistas é o *Intelligent Driving Model with Lane Changing* (IDM_LC) (HÄRRI; FIORE, 2006); e
- *Intersections*: Descreve o tipo de gerenciamento de cruzamentos implementados pelo modelo, se houver. Um dos modelos que gerenciam os cruzamentos é o *Intelligent Driving Model with Intersection Management* (IDM_IM) (HÄRRI; FIORE, 2006).

Três classes de modelos de micro-mobildiade, com um grau crescente de detalhes, podem ser identificados dependendo se o movimento individual dos veículos é obtido por meio das seguintes estratégias:

- De modo determinístico;
- Como uma função do comportamento de veículos nas proximidades de um cenário de pista única;
- Como uma função do comportamento de veículos nas proximidades de um cenário de pistas múltiplas.

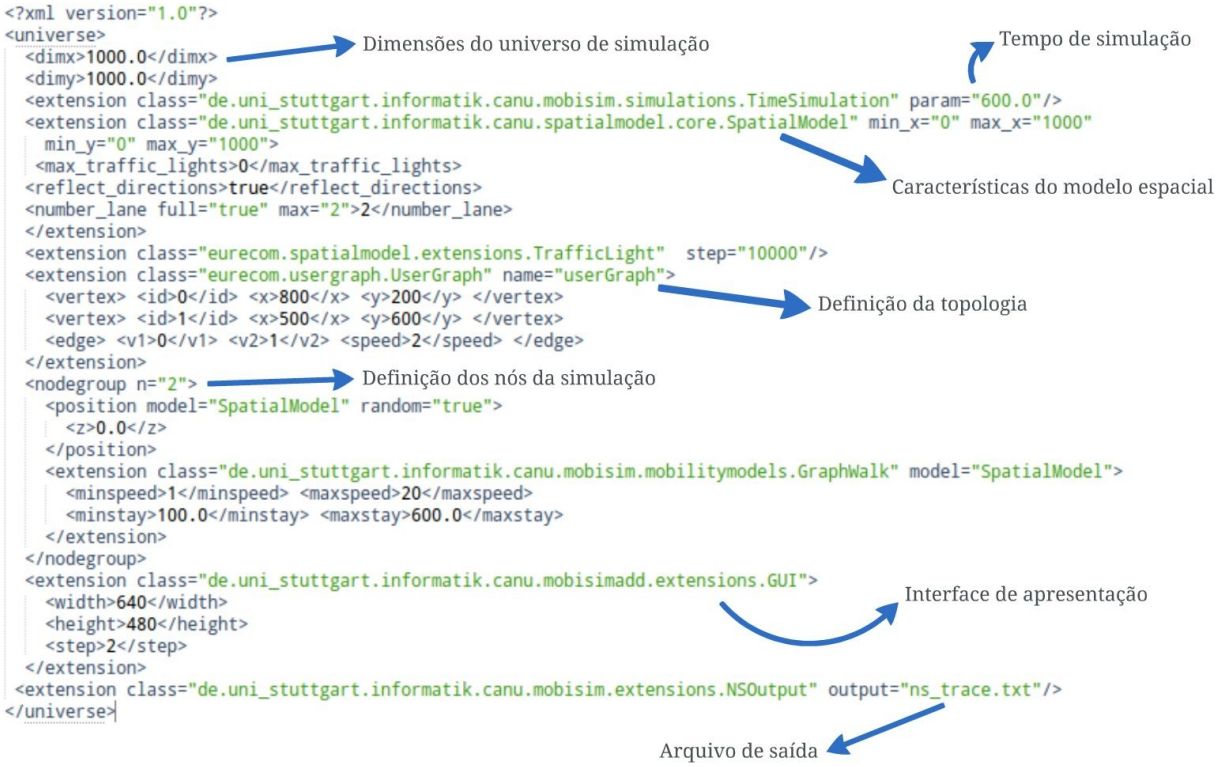
[3]http://www.sciencedirect.com/science/article/pii/0191261581900370

Segundo Fiore (2006), existem diversos modelos matemáticos capazes de representar o possível comportamento dos veículos em estradas. Detalhes sobre esses modelos estão fora do escopo deste trabalho mas podem ser encontrados em (FIORE, 2006).

## 2.2 VANETMOBISIM

VanetMobiSim é uma ferramenta de código aberto utilizada na geração de mobilidade veicular, baseada no simulador CanuMobiSim[4], que foi estendido para fornecer simulações realistas de mobilidade veicular. Este *software* é escrito em Java, deste modo, é independete de plataforma (HÄRRI et al., 2006).

Essa ferramenta utiliza um arquivo no formato XML como parâmetro de entrada em sua execução. Neste documento são definidas todas as configurações realizadas pelo usuário, as quais compõem as diferentes características de mobilidade do cenário simulado. Está apresentado na Figura 4 um exemplo de arquivo XML.




```
<?xml version="1.0"?>
<universe>
  <dimx>1000.0</dimx>
  <dimy>1000.0</dimy>
  <extension class="de.uni_stuttgart.informatik.canu.mobisim.simulations.TimeSimulation" param="600.0"/>
  <extension class="de.uni_stuttgart.informatik.canu.spatialmodel.core.SpatialModel" min_x="0" max_x="1000"
    min_y="0" max_y="1000">
   <max_traffic_lights>0</max_traffic_lights>
  <reflect_directions>true</reflect_directions>
  <number_lane full="true" max="2">2</number_lane>
  </extension>
  <extension class="eurecom.spatialmodel.extensions.TrafficLight"  step="10000"/>
  <extension class="eurecom.usergraph.UserGraph" name="userGraph">
    <vertex> <id>0</id> <x>800</x> <y>200</y> </vertex>
    <vertex> <id>1</id> <x>500</x> <y>600</y> </vertex>
    <edge> <v1>0</v1> <v2>1</v2> <speed>2</speed> </edge>
  </extension>
  <nodegroup n="2">
    <position model="SpatialModel" random="true">
      <z>0.0</z>
    </position>
    <extension class="de.uni_stuttgart.informatik.canu.mobisim.mobilitymodels.GraphWalk" model="SpatialModel">
      <minspeed>1</minspeed> <maxspeed>20</maxspeed>
      <minstay>100.0</minstay> <maxstay>600.0</maxstay>
    </extension>
  </nodegroup>
  <extension class="de.uni_stuttgart.informatik.canu.mobisimadd.extensions.GUI">
    <width>640</width>
    <height>480</height>
    <step>2</step>
  </extension>
 <extension class="de.uni_stuttgart.informatik.canu.mobisim.extensions.NSOutput" output="ns_trace.txt"/>
</universe>
```

**Figura 4: Arquivo XML utilizado para definição do cenário de mobilidade.**

O arquivo apresentado na Figura 4 foi utilizado neste estudo de integração das ferramentas. Detalhes de sua estrutura estão disponíveis na documentação da ferramenta VanetMo-

[4]http://canu.informatik.uni-stuttgart.de/mobisim/

biSim (HÄRRI; FIORE, 2006). Este arquivo define basicamente as seguintes características do cenário de mobilidade:

- Dimensões do cenário: 1.000m x 1.000m;
- Tempo de simulação: 600ms;
- Topologia do grafo utizado para representação do cenário: *User-graph*;
- Características da movimentação através das vias, como velocidades mínima e máxima e tempos mínimo e máximo de parada;
- Número de nós presentes na simulação: 2;
- Apresentação de iterface visual, com dimensões: 640 *pixels* x 480 *pixels*
- Nome do arquivo exportado: ns_trace.txt;

O resultado da execução da ferramenta VanetMobiSim é um arquivo conhecido como *Trace File* que especifica a mobilidade dos veículos que foram simulados, e este arquivo pode variar de acordo com o simulador de rede que irá utilizá-lo. Ao executar a ferramenta uma janela será apresentada para demostrar visualmente a movimentação dos nós no grafo definido, conforme Figura 5. A apresentação desta janela depende da adição da extensão `GUI`[6] no arquivo XML de difinição do cenário.

Após terminada a simulação, o arquivo `ns_trace.txt` terá o conteúdo apresentado pela Figura 6.

É importante observar que o VanetMobiSim suporta tanto descrições de macro-mobilidade, quanto micro-mobilidade para definição de cenários, e isso é fundamental para a qualidade da simulação, ou seja, aproxima-se ao máximo a simulação do cenário real.

### 2.2.1 MACRO-MOBILIDADE UTILIZANDO VANETMOBISIM

Os recursos de macro-mobilidade que podem ser definidos utilizando a ferramenta VanetMobiSim estão dispostas nas próximas subseções.

[6] http://canu.informatik.uni-stuttgart.de/mobisim/

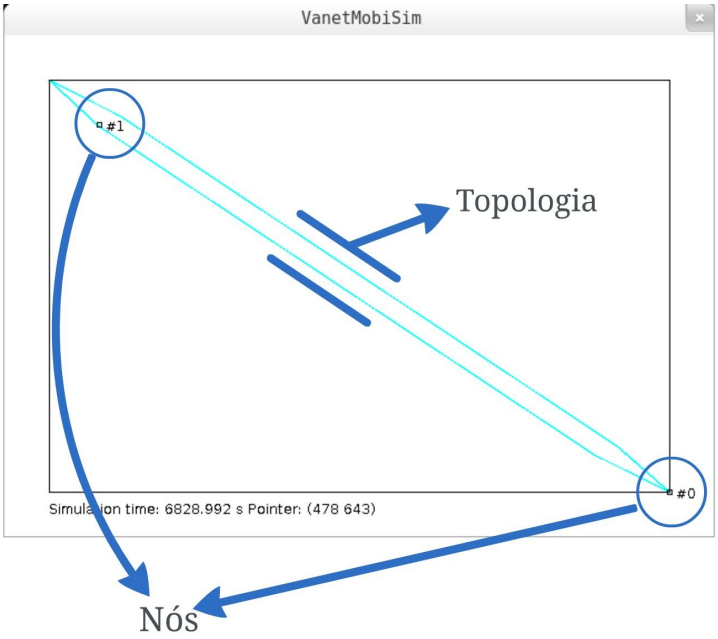



**Figura 5: Exemplo de simulação utilizando a ferramenta VanetMobiSim.**

```
#
# nodes: 2, pause: 3.402823, max speed: 3.402823
# max x = 800.0, max y: 600.0
#
$node_(0) set X_ 764.400001
$node_(0) set Y_ 235.800001
$node_(0) set Z_ 0.0
$node_(1) set X_ 764.400001
$node_(1) set Y_ 235.800001
$node_(1) set Z_ 0.0
$ns_ at 0.0 "$node_(0) setdest 800.000001 200.000001 14.825183"
$ns_ at 0.0 "$node_(1) setdest 800.000001 200.000001 1.7389317"
$ns_ at 29.034 "$node_(1) setdest 775.600001 244.200001 1.7389317"
$ns_ at 282.929 "$node_(1) setdest 535.600001 564.200001 18.663605"
$ns_ at 304.362 "$node_(1) setdest 500.000001 600.000001 18.663605"
$ns_ at 338.061 "$node_(0) setdest 764.400001 235.800001 4.9366074"
$ns_ at 421.03 "$node_(1) setdest 500.000001 600.000001 4.3208804"
$ns_ at 573.057 "$node_(1) setdest 535.600001 564.200001 16.847134"
$ns_ at 576.054 "$node_(1) setdest 775.600001 244.200001 16.847134"
$ns_ at 590.239 "$node_(0) setdest 524.400001 555.800001 7.8125763"
```

**Figura 6: Arquivo ns_trace.txt.**

#### 2.2.1.1 TOPOLOGIAS

As topologias de estradas podem ser definidas pelo usuário (*User-defined*), utilizando arquivos GDF ou TIGER, ou geradas aleatoriamente criando um Diagrama *Voronoi* (*Voronoi Tessellation*) (ZIMMERMANN et al., 2005) de um conjunto de pontos distribuídos de maneira

não uniforme.

Além disso, as seguintes características podem ser difinidas:

- Separação física de tráfegos opostos de cada via;
- Ruas com múltiplas faixas em cada direção;
- Restrição de velocidade em cada segmento de estrada;
- Implementação de sinais de trânsito em cada interseção da estrada.

#### 2.2.1.2 GERAÇÃO DE VIAGEM

A viagem pode ser definida utilizando uma seqüências de atividades, que estabelecem o modo com que os veículos se deslocam de um ponto pré-definido para outro ponto pré-definido, ou aleatoriamente, para veículos que não têm qualquer trajetória específica.

#### 2.2.1.3 SELEÇÃO DE CAMINHO

A seleção de caminho pode acontecer de maneira aleatória ou utilizando o algoritmo de *Dijkstra* (FAN; SHI, 2010) para selecionar o menor caminho. Os custos entre os diferentes pontos podem ser definidos, como as distâncias entre os níveis de congestionamento de tráfego, ou os limites de velocidade.

### 2.2.2 MICRO-MOBILIDADE UTILIZANDO VANETMOBISIM

Os recursos de micro-mobilidade contidos na ferramenta VanetMobiSim são utilizados para simular o comportamento dos motoristas. Dentre estes recuros estão:

- Cálculo da velocidade individual de veículos de uma maneira determinística:
  - *Graph-Based Mobility Model* (GBMM), que propõe um modelo de mobilidade baseado em grafo e que proporciona um modelo de movimentação mais realista se comparado ao modelo aleatório, refletindo adequadamente as restrições espaciais do mundo real (TIAN et al., 2002);
  - *Constant Speed Motion* (CSM), neste modelo a velocidade de cada um dos veículos é determinada com base na posição dos demais e qualquer efeito externo é ignorado (SARMIENTO, 2011); e

  - *Smooth Motion Model* (SMM) (BETTSTETTER, 2001), assim como o CSM, este modelo leva em consideração a presença de veículos nas proximidades para realizar os cálculos de velocidade, e permite de modo adicional que as mudanças de velocidade sejam feitas de maneira suave ao chegar e sair de um ponto (FIORE et al., 2007).

- Cálculo da velocidade dos veículos em função do comportamento de veículos próximos em um cenário de pista única:
  - *Fluid Traffic Model* (FTM) (SESKAR et al., 1992); e
  - *Intelligent Driver Model* (IDM) (TREIBER et al., 2000).

- Gestão de intersecção:
  - *Intelligent Driver Model with Intersection Management* (IDM-IM) (FIORE et al., 2007).

- Mudança de faixa e ultrapassagens:
  - *Intelligent Driver Model with Lane Changes* (IDM-LC) (FIORE et al., 2007).

## 2.3 NETWORK SIMULATOR 3 - (NS-3)

NS-3 (*Network Simulator 3*) é um simulador de rede baseado em eventos "discretos" (apenas eventos que são agendados para executar em determinado tempo de simulação serão realizados), no qual o núcleo da simulação e os modelos são implementados usando a linguagem C++. Ele é construído como uma biblioteca, associada a um programa principal escrito em C++, que define a topologia da simulação e inicia o simulador. NS-3 também exporta boa parte de toda sua API para a linguagem Python, permitindo que os programas desenvolvidos em Python possam importar um módulo "NS3" da mesma maneira que a biblioteca NS-3 está ligada aos executáveis escritos em C++ (NS3, 2010).

O simulador mantém o controle de uma série de eventos que estão programados para executar em um tempo de simulação específico. O trabalho do simulador é executar os eventos em ordem cronológica sequencial. Uma vez que a execução de um evento ocorrer, o simulador realizará o próximo evento (ou vai encerrar a execução se não houverem mais eventos na fila de eventos programados). Se, por exemplo, um evento agendado para o tempo de simulação igual a 100 segundos é executado e nenhum outro evento está programado até o segundo 200, o simulador irá saltar imediatamente de 100 segundos para 200 segundos (tempo de simulação)

para executar o próximo evento. Essa característa define o simulador como sendo de “eventos discretos”.

O NS-3 pretende ser uma nova alternativa ao popular *Network Simulator* 2 (ns-2). Seu desenvolvimento começou em Julho de 2006 e vem sendo aprimorado desde então, contando com a contribuição de vários desenvolvedores. O projeto é financiado por várias instituições como a Universidade de Washington, o Georgia Intitute of Tecnology e o ICSI Center for Internet Research, tendo ainda o suporte do Planète research group do INRIA Sophia-Antipolis (NS3, 2010).

É apresentada na Figura 7 a organização do software NS-3. Detalhes sobre cada uma das parte integrantes da ferramenta, incluindo tutoriais de como utilizá-la, estão disponíveis na página[7] do *software* na *Web* (NS3, 2010).

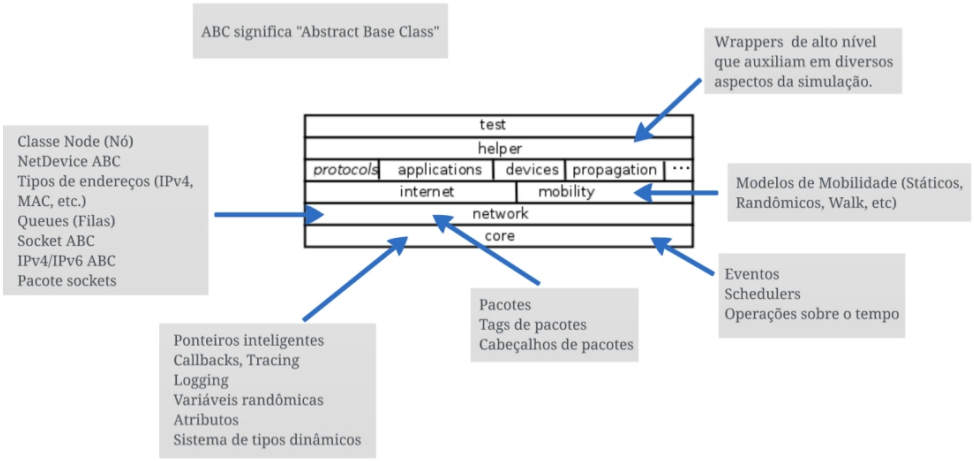



**Figura 7: Organização do *software* NS-3.**

**Fonte: Adaptado de (NS3, 2010)**

Destaca-se na estrutura apresentada na Figura 7 o núcleo do simulador, pois agrega componentes que são comuns em todos os protocolos, tipos de hardware, e modelos ambientais. Sua implementadação está contida no módulo `Core`. Outro importante aspecto é a existência dos Pacotes (*Packages*), que são objetos fundamentais em um simulador de rede e estão implementados no módulo `Network`. Estes dois módulos de simulação compreendem um núcleo de simulação genérico, que pode ser utilizado por diferentes tipos de redes, e não apenas as redes baseadas na Internet, enquanto que os outros módulos são utilizados em tipos específicos de rede ou em determinados tipos de dispositivos.

Além dos recursos citados, o módulo `Mobility` é essencial para a simulação de redes

[7] http://www.nsnam.org/documentation/

móveis e fundamental para o estudo realizado. Nele estão contidos recursos necessários para a simulação de mobilidade, como macanismos para controlar e manter a posição e velocidade de um objeto, meios de identificar a mudança de curso de um nó, e classes auxiliares que podem ser utilizadas para adicionar e configurar mobilidade em objetos.

# 3 MATERIAIS E MÉTODOS

Esta seção apresenta a descrição sobre o mecanismo de integração utilizado, bem como as configurações que foram necessárias para chegar ao resultado final.

## 3.1 VISÃO GERAL

O resultado do estudo acerca das possíveis formas de integração entre as ferramentas, foi o projeto e desenvolvimento de uma aplicação capaz de traduzir o conteúdo do arquivo gerado após a execução do *software* VanetMobiSim, em um modelo compatível com o simulador NS-3. Com este arquivo, montado com a formatação adequada, o simulador NS-3 adquire a capacidade de atribuir a mobilidade dos nós simulados na aplicação VanetMobiSim aos nós de seu cenário.

A leitura e interpretação deste arquivo é realizada através da utilização de um dos *Helpers* disponíveis na ferramenta NS-3. O *Helper* mencionado trata-se do `Ns2MobilityHelper`[8]. Este *Helper* é capaz de atribuir aos nós presentes em um cenário simulado na ferramenta NS-3 as características de mobilidade presentes no arquivo gerado pela ferramenta VanetMobiSim. Entretanto, as informações contidas neste arquivo não estão devidamente formatadas, e é justamente neste ponto que a aplicação criada se aplica.

Na Figura 8 é apresentado um diagrama que demonstra o processo proposto realizar a integração das ferramentas.

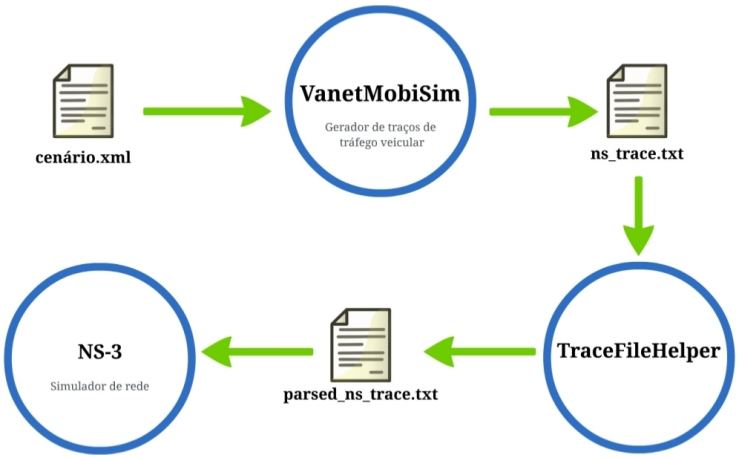



**Figura 8: Fluxo da integração.**

[8] http://www.nsnam.org/doxygen-release/classns3_1_1_ns2_mobility_helper.html

Logo após realizar a simulação do cenário construído na ferramenta VanetMobiSim, utiliza-se a aplicação desenvolvida neste estudo para realizar a tradução do arquivo obtido para um formato adequado ao simulador NS-3.

## 3.2 AMBIENTE DE DESENVOLVIMENTO

O desenvolvimento da aplicativo ocorreu através da utilização da versão 4.7.2 do *Gnu C++ Compile* (g++) em conjunto com o uso da ambiente de desenvolvimento *Eclipse* na versão Juno *Release* 2.

Para a elaboração do diagrama de classes da aplicação, utilizou-se a *Unified Modeling Language* (UML) através da ferramenta Astah Community versão 6.4. Além disso, no estudo realizado, utilizou-se a versão 1.1 do simulador VanetMobiSim, e a versão 3.17 do simulador NS-3.

## 3.3 ESTRUTURA DO APLICATIVO

Para representar a estrutura do aplicativo, foi criado o diagrama de classes. Segundo MARTINS (2007)

> "os diagramas de classe são utilizados para representar a estrutura estática do sistema, composta pelas classes de negócio, classes de interface com o usuário e com outros sistemas, e as classes de controle, responsáveis pelo controle de transações".

O diagrama de classe do aplicativo é apresentado na Figura 9.

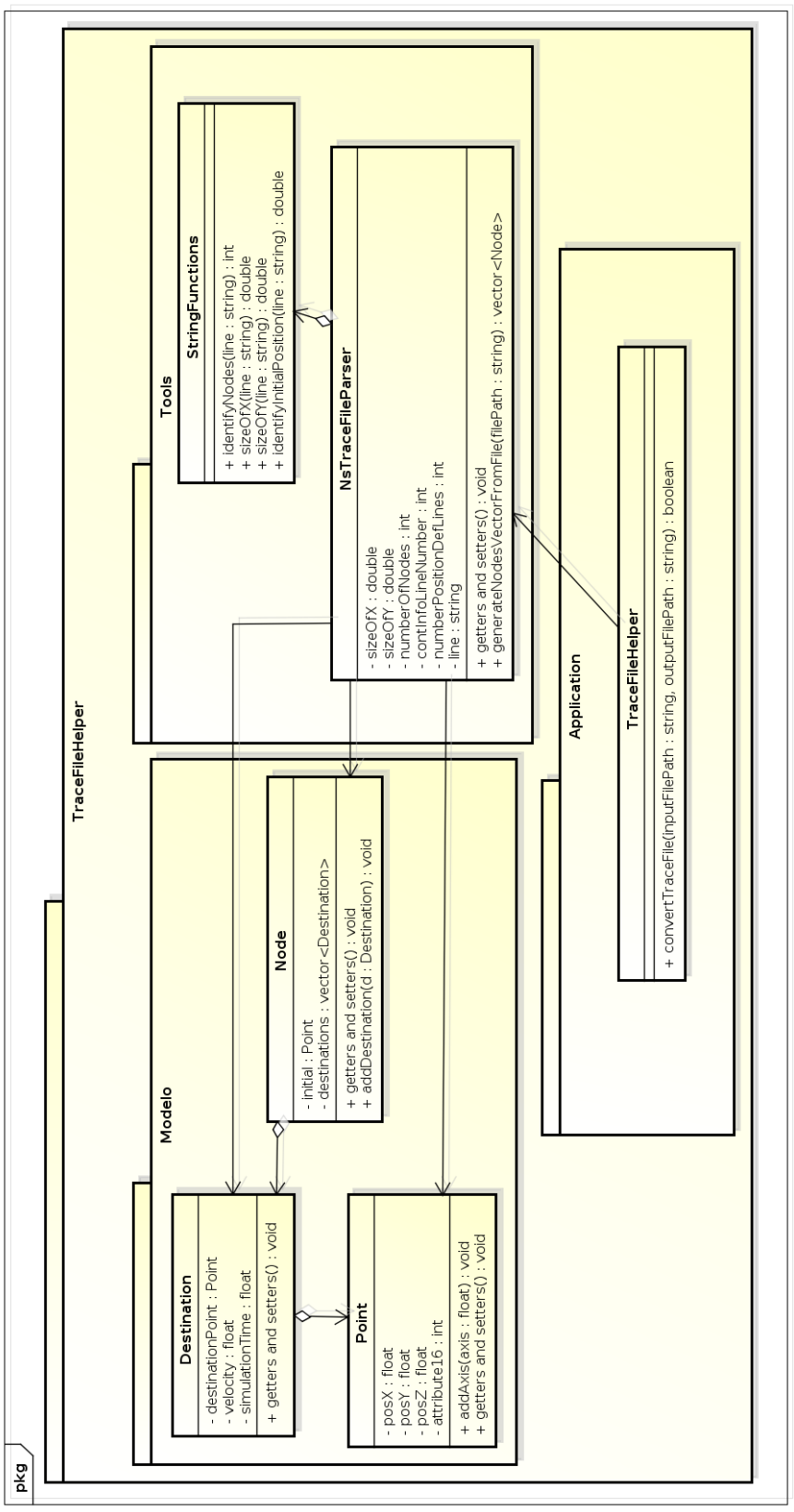



**Figura 9: Diagrama de classes do aplicativo.**

Apresenta-se na Figura 10 a estrutura do projeto do aplicativo criado.

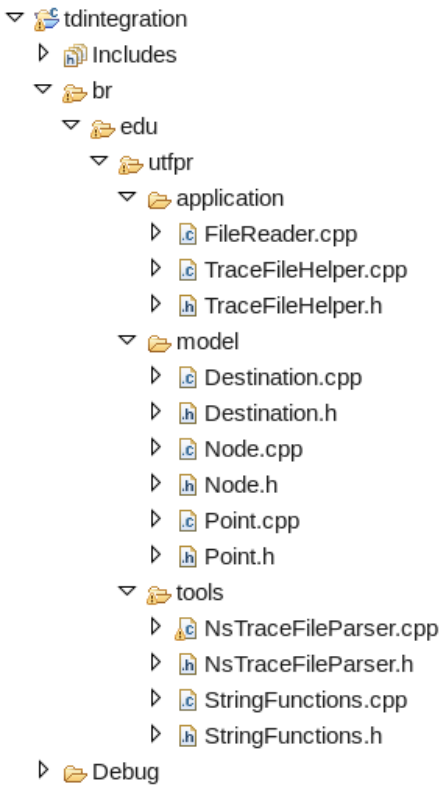



**Figura 10: Estrutura do aplicativo.**

Começando pelas classes contidas no pacote `model` (Modelo), tais classe são utilizadas para abstrair características dos nós contidos nas simulações do aplicativo VanetMobiSim.

A classe `Point` representa as características dos pontos pelos quais um nó (veículo) frequenta durante o tempo de simulação, ou seja, uma instância desta classe agrega em seus atributos as coordenadas no plano cartesiano por onde um nó esteve. O código da classe `Point` é apresentado na Figura 11.

```
#include <cstddef>

class Point {
public:
    float posX;
    float posY;
    float posZ;
    Point();
    virtual ~Point();

    /* Getters e Setters foram omitidos */

    /* Inseri valores para os eixos de maneira sequencial */
    void addAxis(float axis) {
        if (this->posX == NULL) { posX = axis;}
          else if (this->posY == NULL) { posY = axis; }
            else { posZ = axis; }
    }
};
```

**Figura 11: Classe Point.**

A classe `Destination` representa as características dos pontos de destino que um nó recebe durante a execução da simulação, ou seja, um nó parte de uma ponto inicial no tempo zero e, ao longo do tempo de simulação, percorrerá uma coleção de pontos de destino com uma determinada velocidade. Essa classe é apresentada na Figura 12.

```
#include "Point.h"

class Destination {
public:
    Destination();
    virtual ~Destination();
    Point destinationPoint;
    float velocity;
    float simulationTime;

    /* Getters e Setters foram omitidos */

};
```

**Figura 12: Classe Destination.**

A classe `Node` abstrai as características dos nós presentes na simulação, onde cada instância desta classe possui um ponto de inicial e uma coleção de destinos, deste modo, sabe-se o ponto de partida e todos os pontos de destino percorridos por um veículo. Essa classe é apresentada na Figura 13.

```
#include <vector>
#include <iostream>
#include <sstream>
#include <iterator>
#include <string>
#include "Point.h"
#include "Destination.h"
using namespace std;

class Node {
public:
    Node();
    virtual ~Node();
    Point initial;
    vector<Destination> destinations;

    /* Getters e Setters foram omitidos */

    /* Inseri novos destinos no vetor */
    void addDestination(const Destination& destination) {
        destinations.push_back(destination);
    }

};
```

**Figura 13: Classe Node.**

As classes presentes no pacote `Tools` (Ferramentas) são responsáveis por realizar a leitura do arquivo gerado pelo simulador de mobilidade VanetMobiSim e realizar a conversão para o modelo orientado a objetos.

A classe `StringFunctions` é responsável por fornecer métodos para captar informações essenciais das linhas do arquivo gerado, como o número de nós da simulação, os pontos de partida de cada nó e todos os pontos de destino. Essa classe é apresentada na Figura 14.

```
#include <string>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
using namespace std;

class StringFunctions {
public:
    StringFunctions();
    virtual ~StringFunctions();

    /* Identifica a quantidade de nós na simulação */
    static int identifyNodes(string line) {
        return atoi(line.substr(9, (line.find(",") - 9)).c_str());
    }

    /* Identifica o tamanho do eixo X do cenário de simulação */
    static double sizeOfX(string line) {
        return atof(line.substr(line.find("max x") + 8,(line.rfind(",
           ") - (line.find("max x") + 8))).c_str());
    }

    /* Identifica o tamanho do eixo Y do cenário de simulação */
    static double sizeOfY(string line) {
        return atof(line.substr(line.find("max y") + 7,(line.length()
            - (line.find("max y") + 6))).c_str());
    }

    /* Identifica o ponto inicial de um nó */
    static double identifyInitialPosition(string line) {
        return atof( line.substr(line.rfind(" ") + 1,(line.rfind(" ")
            + 1) - line.length()).c_str());
    }
};
```

**Figura 14: Classe StringFunctions.**

A classe `NsTraceFileParser` é responsável por gerar uma coleção de nós a partir da leitura do arquivo de saída da ferramenta VanetMobiSim. Para isso, esta classe utiliza os recursos disponíveis estaticamente em `StringFunctions`. Essa classe é apresentada nas Figuras 15 e 16.

O método `generateNodesVectorFromFile(string filePath)` presente na classe `NsTraceFileParser`, é encarregado de receber o caminho para o arquivo emitido pela ferramenta VanetMobiSim e construir uma coleção de objetos do tipo `Node`. Esta coleção de objetos é utilizada na construção de um arquivo passível de leitura pela ferramenta NS-3.

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <stdio.h>
#include <string.h>
#include <iterator>
#include <vector>
#include "StringFunctions.h"
#include "../model/Point.h"
#include "../model/Node.h"
using namespace std;

class NsTraceFileParser {
private:
    double sizeOfX;
    double sizeOfY;
    int numberOfNodes;
    int contInfoLineNumber = 0;
    int numberPositionDefLines;
    string line;
public:
    NsTraceFileParser();
    virtual ~NsTraceFileParser();

    /* Método responsável por criar um coleção de objetos Node */
    vector<Node> generateNodesVectorFromFile(string filePath) {
        vector<Node> nodes;
        ifstream myfile(filePath.c_str());

        if (myfile.is_open()) {

             while (!myfile.eof()) {
                getline(myfile, line);
                contInfoLineNumber++;

                if (contInfoLineNumber == 2) {
                  numberOfNodes = StringFunctions::identifyNodes(line
                     );
                  sizeOfX = StringFunctions::sizeOfX(line);
                  sizeOfY = StringFunctions::sizeOfY(line);
                  numberPositionDefLines = numberOfNodes * 3;
                    for (int var = 0; var < numberOfNodes; ++var) {
                        Node n;
                        Point p;
                        n.setInitial(p);
                        nodes.push_back(n);
                    }
                }
```

**Figura 15: Classe NsTraceFileParser.**

```
                if (contInfoLineNumber > 3
                 && contInfoLineNumber < (numberPositionDefLines + 4)
                    ) {
                    int pos = atoi(line.substr(line.find("(") + 1,
                       line.find(")")-(line.find("(") + 1)).c_str());
                    nodes.at(pos).initial.addAxis(StringFunctions::
                       identifyInitialPosition(line));
                }

                if (contInfoLineNumber >= numberPositionDefLines + 4)
                    {
                    stringstream ss;
                    ss << line;
                    string ps1, ps2, ps3, ps4, ps5, ps6, ps7, ps8;
                    ss >> ps1 >> ps2 >> ps3 >> ps4 >> ps5 >> ps6 >>
                       ps7 >> ps8;
                    ps4.erase(0, 1);
                    ps8.erase(ps8.length() - 1, 1);
                    int pos = atoi( ps4.substr(ps4.find("(") + 1,
                       ps4.find(")") - (ps4.find("(") + 1)).c_str());
                    Point p;
                    p.setPosX(atof(ps6.c_str()));
                    p.setPosY(atof(ps7.c_str()));
                    p.setPosZ(0);
                    Destination d;
                    d.setDestinationPoint(p);
                    d.setVelocity(atof(ps8.c_str()));
                    d.setSimulationTime(atof(ps3.c_str()));
                    nodes.at(pos).addDestination(d);
                }
            }
            myfile.close();
        } else {
            cout << "Unable to open file" << endl;
        }
        return nodes;
    }
};
```

**Figura 16: Classe NsTraceFileParser (Continuação).**

As classes presentes no pacote `application` (Aplicação) são responsáveis por realizar a conversão do arquivo emitido pelo simulador de mobilidade VanetMobiSim em um arquivo compatível com o formato compreensível pelo simulador de rede NS-3.

A classe `TraceFileHelper` disponibiliza um metódo capaz de realizar a conversão de tal arquivo, e a classe `FileReader` utiliza tal recurso para, de fato, transformar o arquivo em um documento útil ao simulador NS-3. Essa classe é apresentada nas Figuras 17 e 18.

```
1  #include <iostream>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <stdio.h>
5  #include <string.h>
6  #include <fstream>
7  #include <iomanip>
8  #include <sstream>
9  #include <string>
10 #include <string.h>
11 #include "../model/Node.h"
12 #include "../model/Point.h"
13 #include "../model/Destination.h"
14 #include "../tools/NsTraceFileParser.h"
15 using namespace std;
16
17 class TraceFileHelper {
18 public:
19     TraceFileHelper();
20     virtual ~TraceFileHelper();
21
22     /* Método que realiza a conversão dos arquivos */
23     bool convertTraceFile(string inputPath, string outputFile) {
24
25         FILE *arquivo;
26         NsTraceFileParser ns;
27         vector<Node> nodes;
28         nodes = ns.generateNodesVectorFromFile(inputPath);
29
30         if ((arquivo = fopen(outputFile.c_str(), "w")) == NULL) {
31             printf("Erro ao abrir arquivo!!!\n\n");
32             exit(1);
33         }
34
35         for (int var = 0; var < nodes.size(); ++var) {
36             stringstream ss;
```

**Figura 17: Classe TraceFileHelper.**

```
            ss << "$node_(" << var << ") set X_ " << std::
               setprecision(17)
                    << nodes.at(var).initial.getPosX() << endl;
            ss << "$node_(" << var << ") set Y_ "
                    << nodes.at(var).initial.getPosY() << endl;
            ss << "$node_(" << var << ") set Z_ "
                    << nodes.at(var).initial.getPosZ() << endl;

            fputs(ss.str().c_str(), arquivo);
            ss.clear();

            for (int var2 = 0; var2 < nodes.at(var).getDestinations()
               .size();
                    ++var2) {
                stringstream ss2;
                ss2 << "$ns_ at "<< std::setprecision(17)
                        << nodes.at(var).getDestinations().at(var2).
                           getSimulationTime()
                        << " \"$node_(" << var << ") setdest "
                        << nodes.at(var).getDestinations().at(var2).
                           getDestinationPoint().getPosX()
                        << " "
                        << nodes.at(var).getDestinations().at(var2).
                           getDestinationPoint().getPosY()
                        << " "
                        << nodes.at(var).getDestinations().at(var2).
                           getVelocity()
                        << "\"" << endl;
                fputs(ss2.str().c_str(), arquivo);
                ss2.clear();
            }
        }
        return true;
    }
};
```

**Figura 18: Classe TraceFileHelper (Continuação).**

O método `convertTraceFile(string inputPath, string outputFile)` se encerraga de receber o caminho de localização do arquivo de saída gerado pelo simulador VanetMobiSim (parâmetro `inputPath`) e o caminho para o arquivo que será gerado (parâmetro `outputFile`).

A classe `FileReader` é responsável por utilizar uma instância da classe `TraceFileHelper` para realizar a conversão de um arquivo. Essa classe é apresentada na Figura 19.

```
#include "TraceFileHelper.h"
using namespace std;
int main() {
    TraceFileHelper helper;
    helper.convertTraceFile(
            "caminho/inputPath.txt",
            "caminho/outputFile.txt");
}
```

**Figura 19: Classe FileReader.**

# 4 RESULTADOS E DISCUSSÕES

Este capítulo apresenta os resultados obtidos por meio dos testes de integração das ferramentas, validando a solução proposta mediante a exportação de um cenário via VanetMobiSim, utilização da aplicação criada para conversão do arquivo exportado e importação do arquivo gerado para o simulador NS-3.

## 4.1 VALIDAÇÃO DA INTEGRAÇÃO

A simulação realizada nesta seção utiliza o arquivo XML de definição de cenário apresentado na Figura 4. A simplicidade do cenário escolhido se justifica pelo fato de que a quantidade de nós presentes na simulação não interfere no resultado esperado.

É apresentado na Figura 20 o comando de execução do simulador VanetMobiSim, utilizando o arquivo XML como parâmetro de entrada. Maiores detalhes de como utilizar os recursos deste *software* podem ser econtrados na página[9] de seu desenvolvedor na *Web*.


**Figura 20: Comando de execução da ferramenta VanetMobiSim.**

Espera-se com a execução deste cenário, obter uma arquivo igual ao apresentado pela Figura 6. Este arquivo será utilizado pela aplicação desenvolvida, no processo de geração de um novo arquivo, contendo as características exigidas pelo *Helper* do simulador NS-3.

Após execução do cenário na ferramenta VanetMobiSim e obtenção do arquivo de saída, nomeado no exemplo de `ns_trace.txt`, utiliza-se a aplicação desenvolvida para gerar o arquivo adequadamente formatado. Este processo é realizado a partir da execução da classe `FileReader`, apresentada na Figura 19, atribuindo aos parâmetro do método `convertTraceFile`, disponíveis na instância da classe `TraceFileHelper`, os valores dos respectivos arquivos de

[9] http://vanet.eurecom.fr/VanetMobiSim_1_0__Manual.pdf

entrada e saída.

Caso o processo ocorra normalmente, o arquivo apresentado pela Figura 21 é obtido.

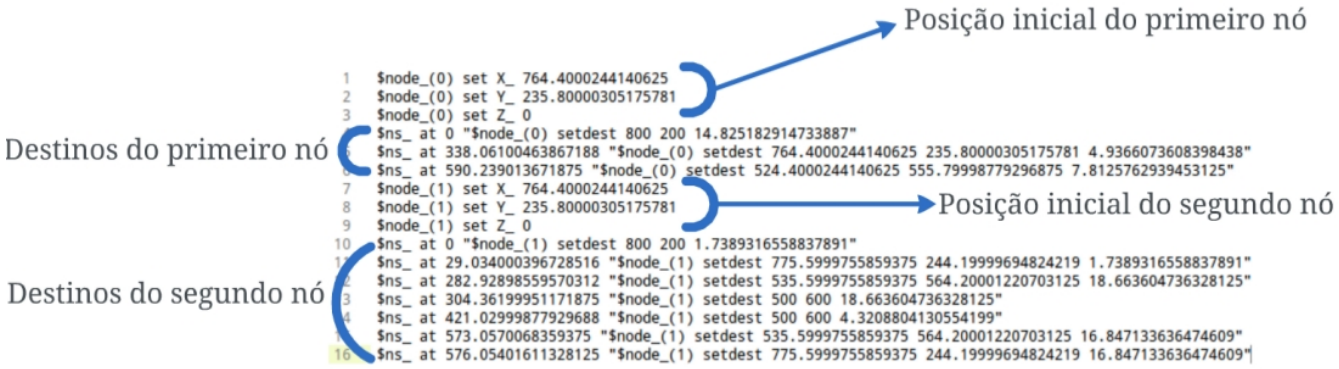



**Figura 21: Arquivo resultante do processo de conversão.**

Após ter simulado o cenário na ferramenta VanetMobiSim e ter utilizado a aplicação desenvolvida para formatar o arquivo gerado, faz-se a utilização do arquivo formatado em um cenário de simulação de rede no *software* NS-3.

O cenário de exemplo que será executado no NS-3 é apresentado nas Figuras 22 e 23. Este arquivo recebeu o nome de `mobi.cc`.

A execução do cenário criado ocorre seguindo o comando apresentado pelo Figura 24. A formatação deste comando deve seguir a definição realizada entre as linhas 39 e 44, apresentadas na Figura 22.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/core-module.h"
#include "ns3/mobility-module.h"
#include "ns3/ns2-mobility-helper.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/point-to-point-layout-module.h"
#include "ns3/netanim-module.h"
using namespace ns3;

// Imprime posição real e velocidade quando um evento de mudança de
   curso
//ocorre
static void
CourseChange (std::ostream *os, std::string foo, Ptr<const
MobilityModel> mobility)
{
  Vector pos = mobility->GetPosition (); // Get position
  Vector vel = mobility->GetVelocity (); // Get velocity

  // Imprime posiç es  e velocidades
  *os << Simulator::Now () << " POS: x=" << pos.x << ", y=" << pos.y
      << ", z=" << pos.z << "; VEL:" << vel.x << ", y=" << vel.y
      << ", z=" << vel.z << std::endl;
}

int main (int argc, char *argv[])
{

  // Habilita Logging para Ns2MobilityHelper
  LogComponentEnable ("Ns2MobilityHelper",LOG_LEVEL_DEBUG);

  // Parser de linha comando
  CommandLine cmd;
  cmd.AddValue ("traceFile", "Ns2 movement trace file", traceFile);
  cmd.AddValue ("nodeNum", "Number of nodes", nodeNum);
  cmd.AddValue ("duration", "Duration of Simulation", duration);
  cmd.AddValue ("logFile", "Log file", logFile);
  cmd.Parse (argc,argv);
```

**Figura 22: Cenário simulado na ferramenta NS-3.**

```
  NodeContainer wifiStaNodes;
  // Criação dos nós
  wifiStaNodes.Create (nodeNum);
  // Criação de Ns2MobilityHelper com arquivo mobilidade predefinido
  Ns2MobilityHelper ns2 = Ns2MobilityHelper (traceFile);

  // Abre log para impressão de sa da
  std::ofstream os;
  os.open (logFile.c_str ());

  // configura os movimentos para cada nó enquento l   o arquivo
     trace file
  ns2.Install ();

  // Configurar callback para registro
  Config::Connect ("/NodeList/*/$ns3::MobilityModel/CourseChange",
                   MakeBoundCallback (&CourseChange, &os));

  Simulator::Stop (Seconds (duration));
  Simulator::Run ();
  Simulator::Destroy ();

  os.close (); // fecha arquivo de Log
  return 0;
}
```

**Figura 23: Cenário simulado na ferramenta NS-3 (Continuação).**

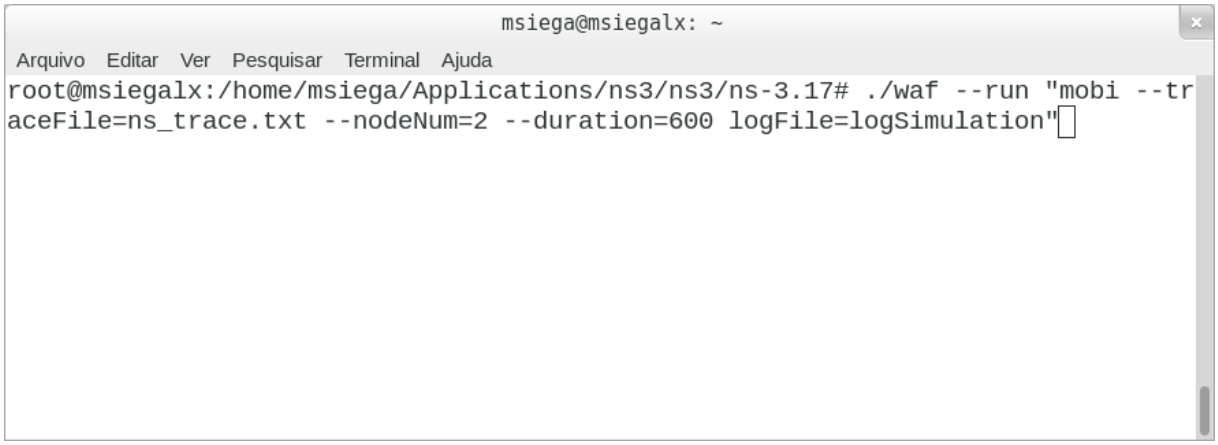



**Figura 24: Comando para simulação de cenário no NS-3.**

O resultado da simulação pode ser observado no arquivo de *Log* gerado pela simulador NS-3, e que recebe o nome dado ao parâmetro `logFile` utilizado no exemplo, ou através dos *logs* apresentados no aplicativo Terminal utilizado para iniciar a simulação. Os *logs* apresentados no aplicativo Terminal estão identificados na Figura 25.

```
1  X=764.4
2  Positions after parse for node 0 0 position = 764.4:0:0
3  Y=235.8
4  Positions after parse for node 0 0 position = 764.4:235.8:0
5  Z=0
6  Positions after parse for node 0 0 position = 764.4:235.8:0
7  X=764.4
8  Positions after parse for node 1 1 position = 764.4:0:0
9  Y=235.8
10 Positions after parse for node 1 1 position = 764.4:235.8:0
11 Z=0
12 Positions after parse for node 1 1 position = 764.4:235.8:0
13 at=0 time=3.40553
14 Calculated Speed: X=10.4536 Y=-10.5123 Z=0
15 Positions after parse for node 0 0 position =800:200:0
16 at=338.061 time=10.2272
17 Calculated Speed: X=-3.48092 Y=3.50047 Z=0
18 Positions after parse for node 0 0 position =764.4:235.8:0
19 at=590.239 time=51.1995
20 Calculated Speed: X=-4.68755 Y=6.25006 Z=0
21 Positions after parse for node 0 0 position =524.4:555.8:0
22 at=0 time=29.0337
23 Calculated Speed: X=1.22616 Y=-1.23305 Z=0
24 Positions after parse for node 1 1 position =800:200:0
25 at=29.034 time=29.0337
26 Calculated Speed: X=-0.840403 Y=1.52237 Z=0
27 Positions after parse for node 1 1 position =775.6:244.2:0
28 at=282.929 time=21.4321
29 Calculated Speed: X=-11.1982 Y=14.9309 Z=0
30 Positions after parse for node 1 1 position =535.6:564.2:0
31 at=304.362 time=2.70514
32 Calculated Speed: X=-13.1601 Y=13.2341 Z=0
33 Positions after parse for node 1 1 position =500:600:0
34 at=421.03 time=0
35 Positions after parse for node 1 1 position =500:600:0
36 at=573.057 time=2.99681
37 Calculated Speed: X=11.8793 Y=-11.946 Z=0
38 Positions after parse for node 1 1 position =535.6:564.2:0
39 at=576.054 time=23.7429
40 Calculated Speed: X=10.1083 Y=-13.4777 Z=0
41 Positions after parse for node 1 1 position =775.6:244.2:0
```

**Figura 25: Arquivo de Log.**

Os *Logs* apresentados pela Figura 25 demonstram que os nós simulados na ferramenta NS-3 seguiram as características definidas no arquivo gerado pelo simulador VanetMobiSim, pois no decorrer da simulação eles visitaram os pontos estabelecidos no arquivo resultante do processo de conversão.

# 5 CONSIDERAÇÕES FINAIS

Com base no estudo realizado e nos resultados obtidos a partir da aplicação desenvolvida neste trabalho, pode-se chegar a algumas conclusões e sugestões para pesquisas futuras.

## 5.1 CONCLUSÃO

A estudo desenvolvido neste trabalho poderá certamente auxiliar os estudantes de redes VANETs a simular cenários sem a necessidade de excessivos gastos com a implantação física destas redes. Outro importante ponto observado na realização deste trabalho é a adição de novos recursos ao simulador NS-3, tornando-o capaz de ser um substituto autossuficiente do simulador NS-2.

As dificuldades encontradas no desenvolvimento do estudo estão relacionadas a incompatibilidade das linguagens em que as ferramentas foram escritas, o que aumenta a dificuldade de integrá-las. Além da questão de compatibilidade, o tempo dedicado ao estudo não foi suficiente, quanto o desejado, para conhecer totalmente as ferramentas, pois os recursos disponíveis em ambas são vastos.

## 5.2 TRABALHOS FUTUROS

Visando melhorar a solução desenvolvida neste trabalho, sugere-se que seja desenvolvido uma classe `Helper`, extendendo a ferramenta NS-3, que utilize as classes de modelo implementadas neste estudo, sem que haja a necessidade de utilizar a formatação de arquivos para compartilhar características de mobilidade.

Sugere-se também a criação de uma ferramenta que seja capaz de representar graficamente a estrutura de uma rede VANET simulada no *software* NS-3, em especial a topologia definida e os nós nela contidos.

# REFERÊNCIAS


AL-KHWILDI, A.; AL-RAWESHIDY, H. A proficient path selection for wireless ad hoc routing protocol. In: **Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference**. [S.l.: s.n.], 2006.

ALVES, R. d. S. et al. Redes veiculares: Principios, aplicações e desafios. In: SBRC (Ed.). **in Minicursos do Simpósio Brasileiro de Redes de Computadores - SBRC'2009**. Brasil: In Minicurso SBRC, 2009. p. 199 –254.

BETTSTETTER, C. Smooth is better than sharp: A random mobility model for simulation of wireless networks. **In MSWiM 2001**, v. 1, p. 19 - 27, 2001.

CAMPOS, C. P. de; FREITAS, E. G. de. **Geometria Computacional**. Dezembro 1998. Disponível em: <http://www.ime.usp.br/∼freitas/gc/index.html>.

CARLINK::UMA. **D2006/6 - Evaluating VANET simulators for CARLINK primary applications.** [S.l.], 2006.

CARLINK::UMA. **D1.3.1-VanetMobiSim: The vehicular mobility model generator tool for CARLINK**. [S.l.], 2007.

FAN, D.; SHI, P. Improvement of dijkstra's algorithm and its application in route planning. In: **in Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on**. Yantai, Shandong: [s.n.], 2010. v. 4, p. 1901 – 1904.

FIORE, M. **Mobility Models in Inter-Vehicle Communications Literature**. Politecnico di Torino, Itália, Novembro 2006.

FIORE, M. et al. Vehicular mobility simulation for vanets. In: **In 40th IEEE Annual Simulation Symposium (ANSS07)**. Norfolk, USA: [s.n.], 2007. v. 1, p. 301– 309.

HÄRRI, J.; FILALI, F.; BONNET, C. **Mobility Models for Vehicular Ad Hoc Networks: A Survey and Taxonomy**. Março 2007. Disponível em: <http://www.comsoc.org/cst/>.

HÄRRI, J. et al. Vanetmobisim: Generating realistic mobility patterns for vanets. In: **VANET'06: Proceedings of the 3rd International Workshop on Vehicular ad hoc Networks**. [S.l.: s.n.], 2006. p. 96 - 97.

HÄRRI, J.; FIORE, M. **VanetMobiSim Vehicular Ad hoc Network mobility extension to the CanuMobiSim framework**. [S.l.], 2006.

LI, F.; WANG, Y. Routing in vehicular ad hoc networks: A survey. In: IEEE. **Vehicular Technology Magazine**. [S.l.], 2007.

LOPES, A. F. R. **Realistic VANET Simulations**. Dissertação (Mestrado) — Instituto Superior Técnico - Departamento de Engenharia e Eletrotécnica e de Computadores, 2011.

MARTINS, J. C. C. **Técnicas para Gerenciamento de Projetos de Software**. [S.l.]: Brasport, 2007.

NS3. **NS3 Manual**. 2010. Disponível em: <http://www.nsnam.org/docs/release/3.17/manual>.

SARMIENTO, A. S. **Multimedia Services and Streaming for Mobile Devices: Challenges and Innovation**. 1. ed. [S.l.]: IGI Global, 2011.

SESKAR, I. et al. Rate of location area updates in cellular systems. **In 42nd Vehicular Technology Conference**, v. 1, p. 694 - 697, 1992.

TIAN, J. et al. Graph-based mobility model for a mobile ad hoc network simulation. In: **In Proceedings of the 35th Annual Simulation Symposium**. [S.l.: s.n.], 2002. v. 1, p. 337 - 344.

TREIBER, M.; HENNECKE, A.; HELBING, D. Congested traffic states in empirical observations and microscopic simulations. ***Physical Review E***, v. 1, p. 62 - 1805, 2000.

ZIMMERMANN, H.-M.; GRUBER, I.; ROMAN, C. A voronoibased mobility model for urban environments. In: **In Proc. of the European Wireless 2005 (EW05)**. Nicosia, Cyprus: [s.n.], 2005. v. 1, p. 4 - 12.