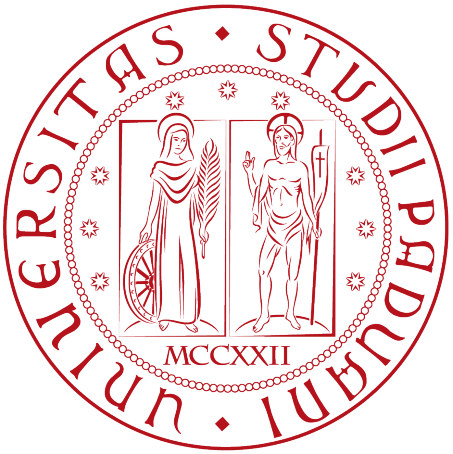
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Java Content Repository per la persistenza di prodotti commerciali

Tesi di laurea triennale

Relatore

Prof. Tullio Vardanega

Laureando

Jordan Gottardo

Anno Accademico 2016-2017

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di trecentoventi ore, dal laureando Jordan Gottardo presso l'azienda IBC S.r.l. di Peraga (PD).

Gli obiettivi principali da raggiungere erano due. In primo luogo, veniva richiesto uno studio degli *standard* JSR 170 e JSR 283, che descrivono le API per l'utilizzo di Java Content Repository (JCR). Era richiesta la produzione di documentazione ed esempi di codice sorgente che sfruttassero tali API.

Il secondo obiettivo riguardava l'implementazione di un prototipo, sottoforma di *web app*, che permettesse la memorizzazione di prodotti commerciali aventi attributi variabili. Era richiesta inoltre l'implementazione di funzionalità di ricerca per effettuare selezioni mirate di prodotti in più passi.

La libreria da utilizzare per la persistenza delle informazioni era Apache Jackrabbit, mentre il *framework* per la realizzazione dell'interfaccia grafica era di libera scelta.

Il presente documento è organizzato in quattro capitoli:

* **L'azienda:** in questo capitolo presento l'azienda che ha ospitato lo *stage*, IBC S.r.l., fornendo descrizioni del contesto aziendale e del modo di lavorare. Descrivo inoltre i prodotti e i servizi che essa offre sul mercato.

* **L'offerta di *stage*:** all'interno di questo capitolo descrivo il progetto di *stage* offerto, soffermandomi sulle motivazioni aziendali e personali che hanno portato a questa scelta. Elencherò inoltre gli obiettivi da raggiungere.

* **Svolgimento del progetto:** in questo capitolo presento le attività svolte durante lo *stage* per il raggiungimento degli obiettivi prefissati.

* **Analisi retrospettiva:** all'interno di questo capitolo fornisco un'analisi retrospettiva sugli obiettivi dello *stage*. Fornisco inoltre una descrizione di alcune mancanze nell'insegnamento accademico che dovrebbero essere aggiunte al piano didattico per un efficace approdo nel mondo del lavoro.

Nel documento utilizzerò le seguenti notazioni tipografiche:

* *Italico*: termine in lingua straniera.

* `Monospace`: nome di *file*, classe o codice sorgente.

* Termine in azzurro: termine a glossario, solo la per la prima occorrenza di ogni capitolo. Cliccando sul termine è possibile leggere la spiegazione.

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Tullio Vardanega, relatore della mia tesi, per l'aiuto fornitomi durante la stesura del documento.

Ringrazio IBC, in particolare Denis Corà e Stefano Gesuato, per il supporto durante il periodo di stage.

Desidero infine ringraziare la mia famiglia per il sostegno che mi ha fornito in questi anni e Giulia, Giovanni P. e Giovanni D. per il lavoro svolto durante la realizzazione del progetto di ingegneria del software.

Padova, Settembre 2017 Jordan Gottardo

Contents

List of Figures

List of Tables

Chapter 1

Tools and applications

In this chapter I will describe the tools and softwares I have utilized to carry out the simulations.

1.1 Obstacle Shadowing propagation loss model

The original thesis **ROM2017** implemented a Radio Propagation Model (RPM)

1.2 Network Simulator 3

Network Simulator 3 (ns-3) is a discrete-event network simulator for Internet systems, targeted primarily for research and educational use. ns-3 is free software, licensed under the GNU GPLv2 license, and is publicly available for research, development, and use.

ns-3 development began in 2006 by a team lead by Tom Henderson, George Riley, Sally Floyd and Sumit Roy. Its first version was released on June 30, 2008.

ns-3 is the successor of ns-2, released in 1989. The fact that the former was built from scratch makes it impossible to have backward compatibility. In fact, ns-2 used oTCL scripting language to describe network topologies and C++ to write the core of the simulation. This choice was due to avoid the very time consuming C++ code recompilation, exploiting the interpreted language oTCL. ns-2 mixed the "fast to run, slow to change" C++ with the "slow to run, fast to change" oTCL language. Since compilation time was not an issue with modern computing capabilities, ns-3 developers chose to utilize exclusively C++ code (and optional Python bindings) to develop simulations.

1.2.1 Modules and module structure

ns-3 is composed of various modules, which are groups of classes, examples and tests each related to a certain feature. The components of a module work in a cohesive way in order to offer APIs to other modules and users. Some examples of built-in modules are:

* WiFi;
* AODV;

* CSMA.

The obstacle shadowing propagation loss model and the Fast Broadcast algorithm have been implemented as modules too.

Modules follow a prototypical structure in order to promote clarity and offer built-in documentation. Figure 1.1 shows the typical module structure.

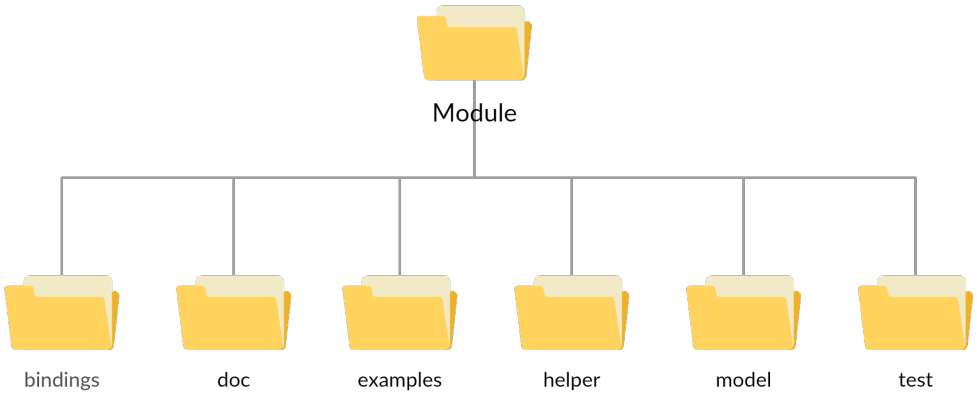


Figure 1.1: ns-3 module structure

The following directories can be found inside a module's root directory:

* **bindings:** Python bindings used to make the module's API compatible with Python;
* **doc:** documentation of the module;
* **examples:** examples and proof of concepts of what can be done using the module;
* **helper:** higher level APIs to make the module easier to use;
* **model:** headers and source files which implement the module's logic;
* **test:** test suite and test cases to test the module.

1.2.2 Key elements

The element at the base of ns-3 is called *node*, instance of `ns3::Node`. A node can be thought of as a shell of a computer. Various other elements can be added to nodes, such as:

* NetDevices (e.g. NICs), which enable nodes to communicate over *channels*;
* protocols;
* applications.

It is the last those which implement the logic of a simulation. For example, the `UdoEchoClientApplication` and `UdpEchoServerApplication` can be used to implement a client/server application which exchange and print the packets' content over

the network. The Fast Broadcast protocol has been implemented as an application as well.

The *channels* model various type of transmission media, such as the wired and the wireless ones.

1.2.3 Structure of a simulation

A simulation can be implemented in many ways, but in most cases the following steps are executed:

* manage command line arguments (e.g. number of nodes to consider in the simulation, transmission range, etc.);
* initialize all the necessary fields in classes;
* create nodes;
* set up physical and MAC layers;
* set up link layer, routing protocols and addresses;
* configure and install applications on nodes;
* position nodes and (optionally) give them a mobility model;
* schedule user defined events, such as transmissions of packets;
* start the simulation;
* collect and manage output data.

1.2.4 NetAnim

Netanim is an offline animator tool based on the Qt toolkit. It collects an XML tracefile during the execution of a simulation and can be used to animate the simulation, analyzing packet transmissions and contents.

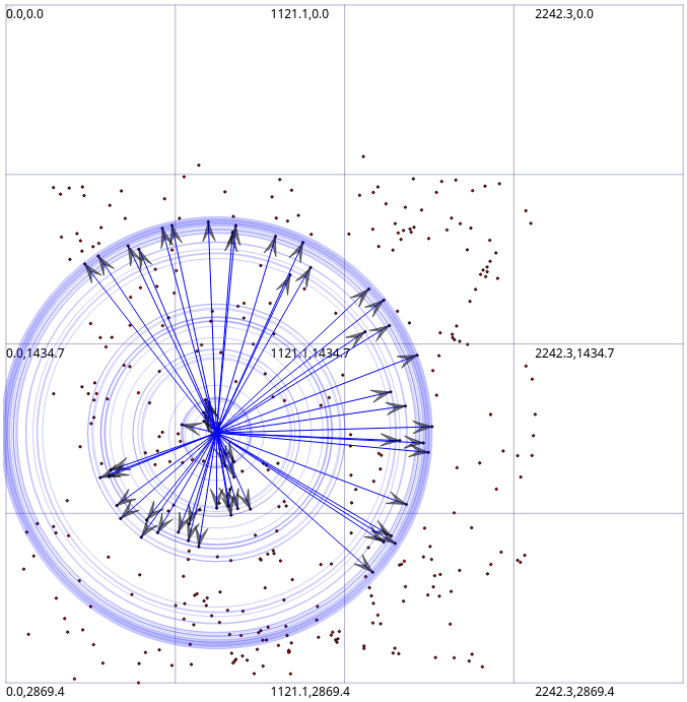


Figure 1.2: Packet transmission in NetAnim

1.3 Simulation of Urban MObility

Simulation of Urban MObility is an open-source road traffic simulation package. It is written in C++ and licensed under GPLv3.

It offers different tools to analyze and manage real maps from the urban mobility point of view, including pedestrian movement and various types of vehicles.

The original work **ROM2017** utilized SUMO to produce, starting from real maps obtained from OpenStreetMap (OSM), two files:

1. a `.poly` file using the SUMO tool Polyconvert. This file contains information about all the obstacles, such as buildings, useful for the Obstacle Shadowing module.

2. a `.ns2mobility` file using the SUMO tool TraceExporter. This file contains information about the vehicles and their positioning.

The process of generating these two files necessary for ns-3 simulations requires some intermediate steps. The full process is represented in figure 1.3.

The original work considered a only a distance of 25 meters between vehicles; this work considers various distances, ranging from 15 to 45 meters. Figure 1.4 shows the same scenario (Padua) with different distances between vehicles.

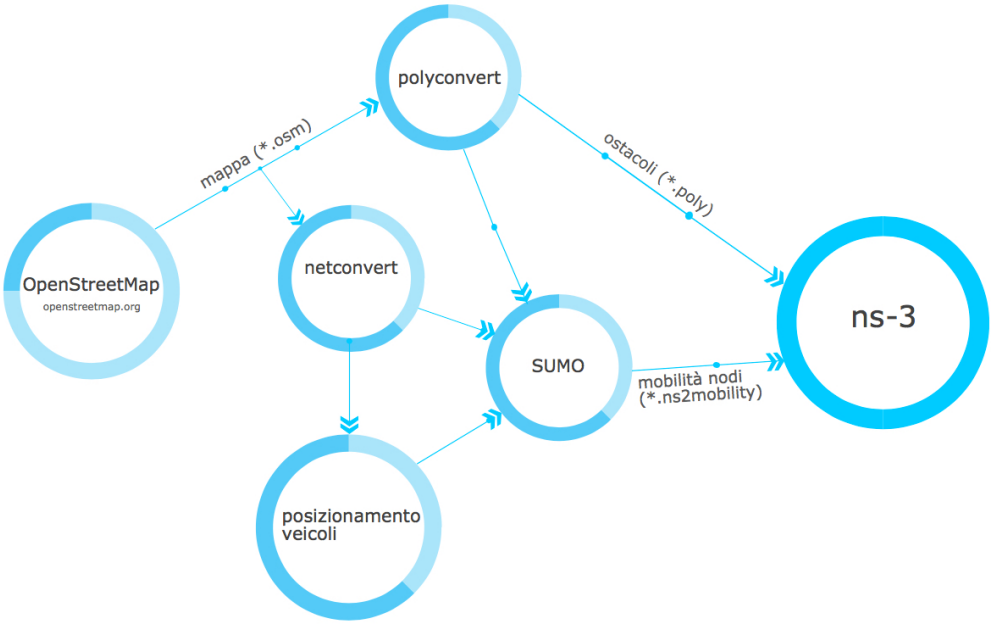


Figure 1.3: Steps to generate necessary files using SUMO (**ROM2017**)

Figure 1.4: Padua scenario with vehicle distance equals to 15 meters (top) and 45 meters (bottom). Vehicles painted yellow

Glossary

Framework È un'architettura logica di supporto (spesso un'implementazione logica di un particolare *design pattern*) su cui un *software* può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore. iii, 5

Jackrabbit Apache Jackrabbit è una libreria Java open source che fornisce un'implementazione di un Java Content Repository, così come definito dagli standard JSR 170 e JSR 283. iii, 5

JSR 170 Java Request Specification rilasciato il 17 giugno 2005. Descrive le API per l'utilizzo di Java Content Repository. È conosciuto anche come "JCR v1.0 Specifications". iii, 5

JSR 283 Java Request Specification rilasciato il 25 settembre 2009. Rispetto a JSR 170, aggiunge (e in alcuni casi rimpiazza) alcune API e funzionalità. È conosciuto anche come "JCR v2.0 Specifications". iii, 5

Web app (ing. applicazione web). È un'applicazione fruibile tramite *web browser*. iii, 5