Alan D. Barroso
Felipe Santana
Victor Lassance

Escola Politécnica da USP

Tutorial for the Creation of a Robocup Rescue Simulator Map

Alan D. Barroso
Felipe Santana
Victor Lassance

Table of contents

1 - Introduction

The Robocup Rescue Simulator comes with some example maps that can be used to test and analyze the agents' performance. Although useful, these different test maps do not provide all necessary scenarios to evaluate a specific strategy or overall performance of the agents in different situations. Furthermore, a team may need to assess their performance picturing a more realistic and familiar scenario, in order to bring a more concrete perspective to the solution.

Therefore, it would really be useful for the different teams to be able to create maps, based on their particular needs and on real world scenarios. In this tutorial, we will provide a step by step procedure one should follow to create a map compatible with the Robocup Rescue Simulator, using a desired area provided in the OpenStreetMap server. We will illustrate each step based on what we did to create the map of University of São Paulo.

2 - Concepts

OpenStreetMap

OpenStreetMap (OSM) is a free worldwide map developed collaboratively by the community, it provides free to use geographic information from many places around the world. It was created under the project Planet.osm, that aims to provide geographic information in one XML file, containing descriptions of Nodes, Ways and Relations. A new version of the project is released every week in a big XML file (around 30GB compressed). One can download the entire map, or the latest changeset, on the following url: http://planet.openstreetmap.org/. The image below shows the layout of a reduced .osm file.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.0.2">
 <bounds minlat="54.0889580" minlon="12.2487570" maxlat="54.0913900" maxlon="12.2524800"/>
 <node id="298884269" lat="54.0901746" lon="12.2482632" user="SvenHRO" uid="46882" visible="true" version="1" changeset="676636"
timestamp="2008-09-21T21:37:45Z"/>
 <node id="261728686" lat="54.0906309" lon="12.2441924" user="PikoWinter" uid="36744" visible="true" version="1" changeset="323878"
timestamp="2008-05-03T13:39:23Z"/>
  <tag k="name" v="Neu Broderstorf"/>
  <tag k="traffic_sign" v="city_limit"/>
 </node>
 ...
 <node id="298884272" lat="54.0901447" lon="12.2516513" user="SvenHRO" uid="46882" visible="true" version="1" changeset="676636"
timestamp="2008-09-21T21:37:45Z"/>
 <way id="26659127" user="Masch" uid="55988" visible="true" version="5" changeset="4142606" timestamp="2010-03-16T11:47:08Z">
  <nd ref="292403538"/>
  <nd ref="298884289"/>
  ...
  <nd ref="261728686"/>
  <tag k="highway" v="unclassified"/>
  <tag k="name" v="Pastower Straße"/>
 </way>
 <relation id="56688" user="kmvar" uid="56190" visible="true" version="28" changeset="6947637" timestamp="2011-01-12T14:23:49Z">
  <member type="node" ref="294942404" role=""/>
  <member type="node" ref="364933006" role=""/>
  <member type="way" ref="4579143" role=""/>
  ...
  <member type="node" ref="249673494" role=""/>
  <tag k="name" v="Küstenbus Linie 123"/>
  <tag k="network" v="VVW"/>
  <tag k="operator" v="Regionalverkehr Küste"/>
  <tag k="ref" v="123"/>
  <tag k="route" v="bus"/>
  <tag k="type" v="route"/>
 </relation>
 ...
</osm>
```

Figure 1 - OSM Layout

JOSM

JOSM is an open Java-based OpenStreetMap editor, an application originally developed by Immanuel Scholz and currently maintained by Dirk Stocker. In this tutorial, we will use JOSM to download a part of the OSM map, as well as to edit the original map to enable conversion.

Geographic Markup Language

Geographic Markup Language (GML) is a XML-based grammar used to describe interchangeable geographic information. It was defined by the Open Geospatial Consortium, and it is largely used for providing a rich set of markup primitives that allow the creation of application specific schemas, such as the definition of Buildings and Roads in our case. The current version of the Robocup Rescue Simulator accepts gml files as map representation.

osm2gml Library

The osm2gml library was created for enabling conversion from OSM to GML standard, transforming the XML file from one format to the other. The conversion process will change the features in the original map, in order to make it compatible with the GML

representation for the maps in the Simulator.It is based on 9 different steps, including Splitting Intersecting Edges and Merging Adjacent Shapes, and its source code is available on the Simulator project folder under **/modules/maps/src.**

3 - Overview

The process of creating the map for the Robocup Rescue Simulator, in a nutshell, is comprised of 3 basic steps: Capture the map, convert the .osm file into gml format, create a valid robocup rescue scenario for the map on the simulator.

In the first step, we will use the JOSM editor to browse through the OSM worldwide map, select the desired area and download the map information for that area into a .osm file. Section 4 contains a more detailed and illustrated explanation on how to proceed on the first step. Then, in the second step we will begin an iterative process, using JOSM to edit the original OSM map, making it convertible to GML, and using the osm2gml library for converting the map into GML format (if the conversion fails, one should retry editing it on JOSM). Section 5 shows more details on how to use the library, as well as some recurrent changes that must be made on the OSM map before converting it. Finally, the last step comprises of the creation of the scenario file for the created GML map, setting the positions for the agents, buildings (Central Agents and Fires) and other features (Gas Stations), according to the entities represented in the .gml file. Section 6 will provide a more detailed explanation on how to setup the scenario.

4 - Map Capturing

The first step to create a map for the Robocup Rescue Simulator is the selection and capture of a desired region on the world map. For that purpose, we are going to use the tool JOSM, that allows us to browse through the OpenStreetMap (see previous section) world map, select a region and download it as a .osm file. The following procedure will show the series of steps to follow in order to capture the map region, using JOSM.

Download and run JOSM

In order to download the tool, go to http://josm.openstreetmap.de/ and select one of the alternatives for using it (Installer, Launcher or Download .jar file). The recommended version is using the Launcher (josm.jnlp), which is compatible with any environment. The image below illustrates the home screen of JOSM tool.

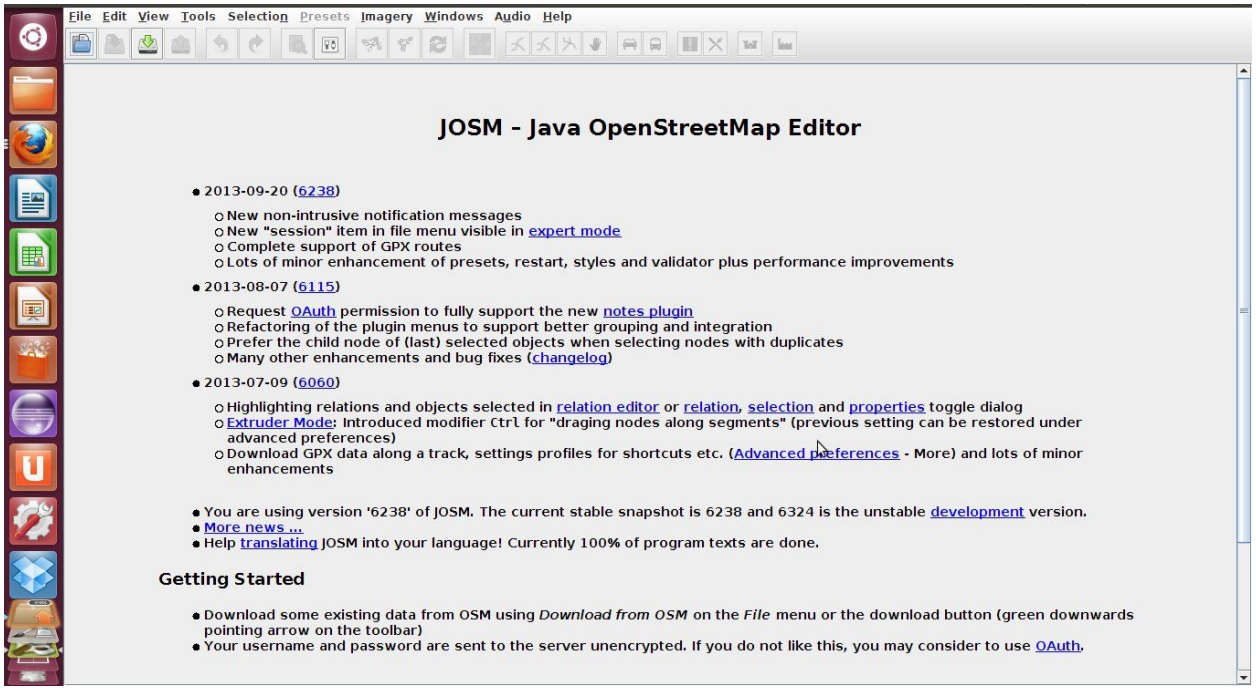

Figure 2 - JOSM Home Screen

Select and download the region of the map

On the left side of the toolbar, you will find a button with a green arrow “Download Map from OSM Server”, click it to start browsing the map or press Ctrl+Shift+Down. The start screen on the map is zoomed out (see figure 3), so you will have to manually zoom in until you find the desired region. After finding the region of the map you need, select the area you want and press enter to start downloading it. Figure 4 shows the selected area corresponding to University Of São Paulo map.

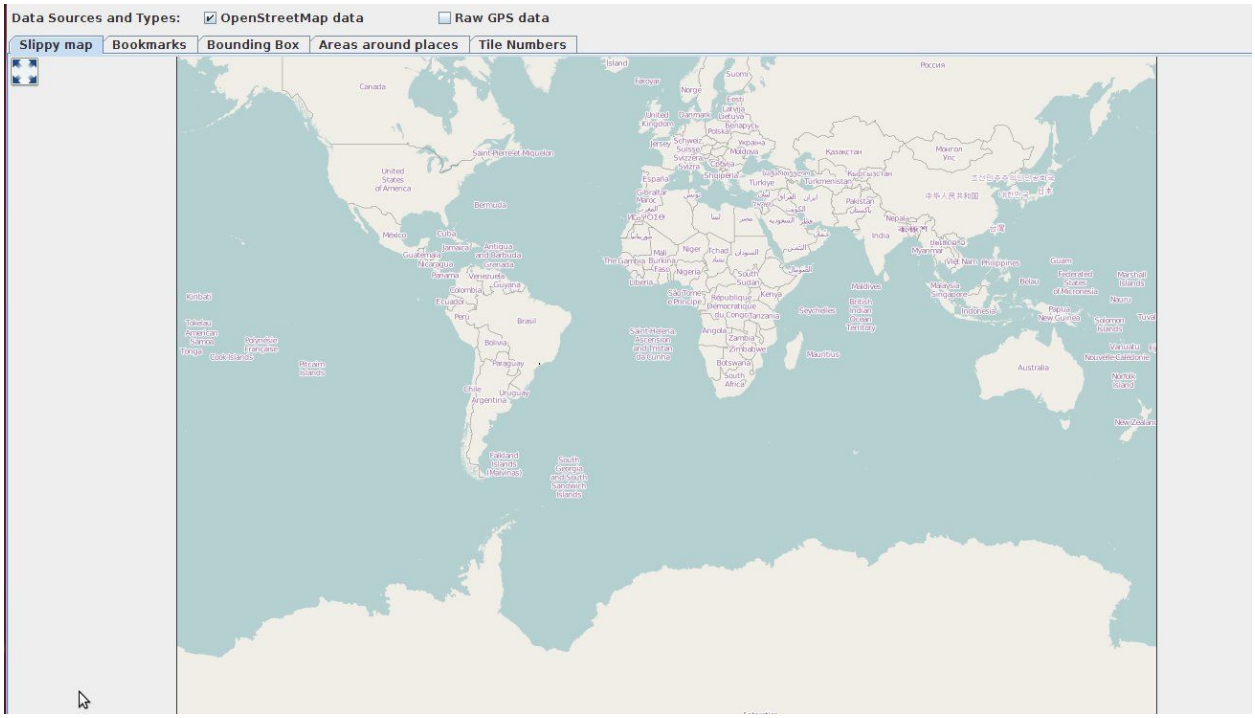

Figure 3 - JOSM Zoomed Out

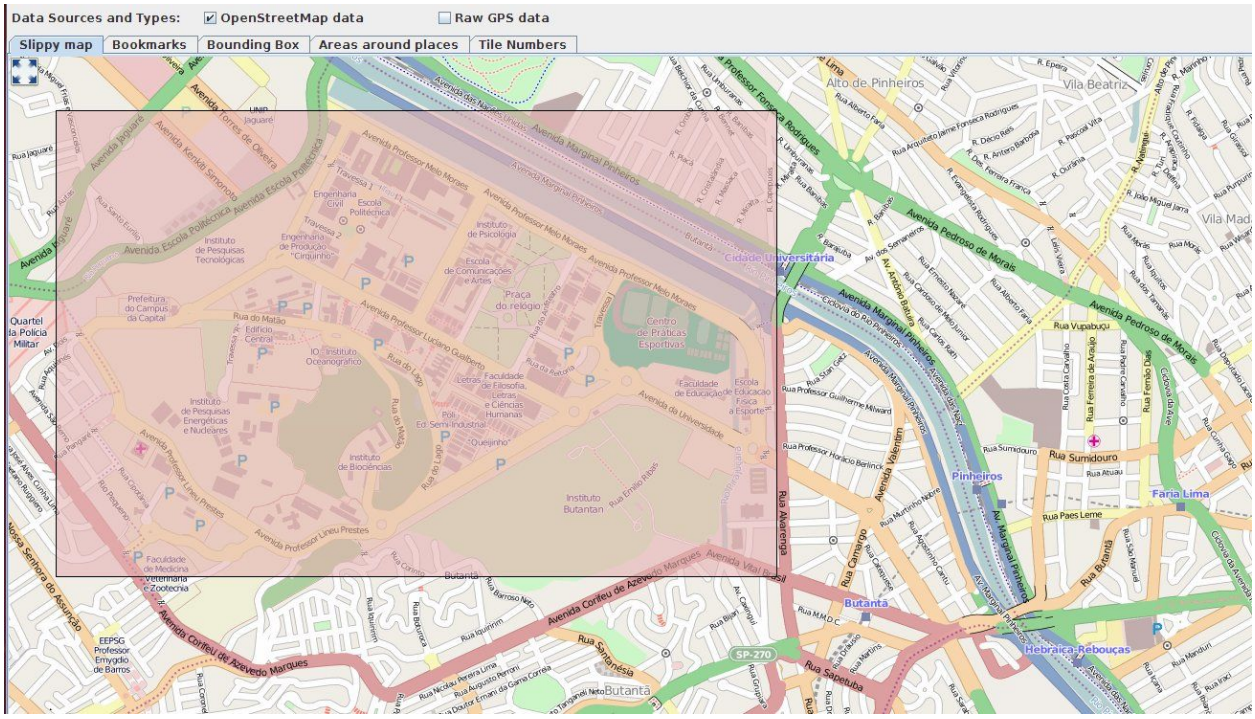

Figure 4 - JOSM Zoomed In USP

Save the OSM file

After JOSM finishes downloading the area of the map from the OpenStreetMap server, it will open the edit screen with the downloaded map on display. Before starting editing it, save the map in a separate folder (File->save as..), see figure below.

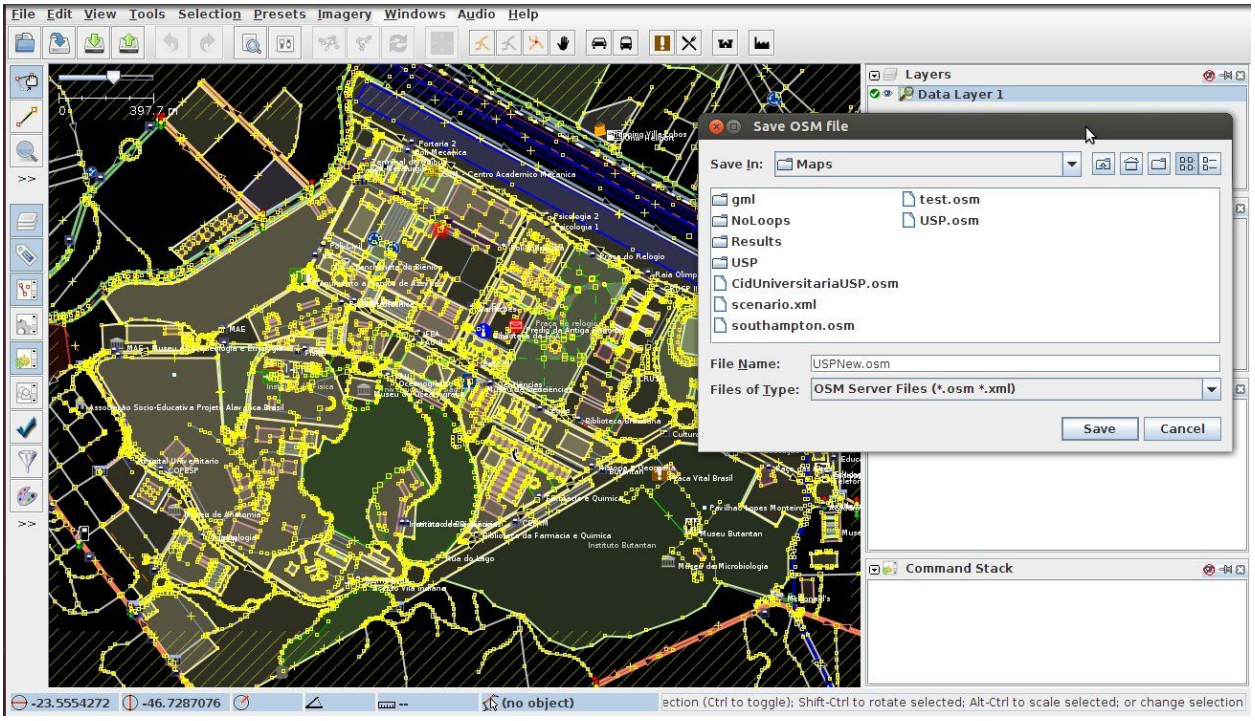

Figure 5 - JOSM Downloaded USP

In these first steps you have used the JOSM tool to browse the world map provided by OpenStreetMap servers, select the desired area and download it to your computer as a .osm file. In the next sections we will show how to edit the map in order to make it convertable to gml format.

5 - Convert the map into GML format

After downloading the map in OSM format, you will now need to convert it into a format that is compatible with the current version of the simulator (GML). In order to do that, you will use a library provided in the code of the Simulator itself, called osm2gml, that will convert your map following a series of steps, such as merging shapes and separating overlapping streets.

However, the original map downloaded from the OSM server will normally contain some kinds of shapes and streets that will cause the conversion to fail, due to an exception risen in the library code. Also, some of the buildings and streets in the original map are not

marked as buildings and streets, which makes them disappear in the conversion process. These are some examples of problems that may prevent the proper conversion of the OSM map; for overcoming them, some tweaks will need to be made, using JOSM. Note that some changes on the library code may be necessary as well.

Compiling the library code

The osm2gml library contains classes that will be used to convert our map (OSM format) into the GML format, compatible with the current version of the simulator. Its source code is located under the folder **/modules/maps/src,** in this tutorial we will compile the library using eclipse, in order to make it easier to modify some pieces of code, if necessary, as well as to be able to debug the application to pinpoint errors on the map. Note that it is also possible to build the library by using ant build in the source folder.

Importing the library source code into an eclipse project

Open eclipse, create a new Java Project, then right click it and select Import->File System. Browse until the src folder of the library code, and click ok. The following image illustrates how to import the library source code into a project on the workspace.

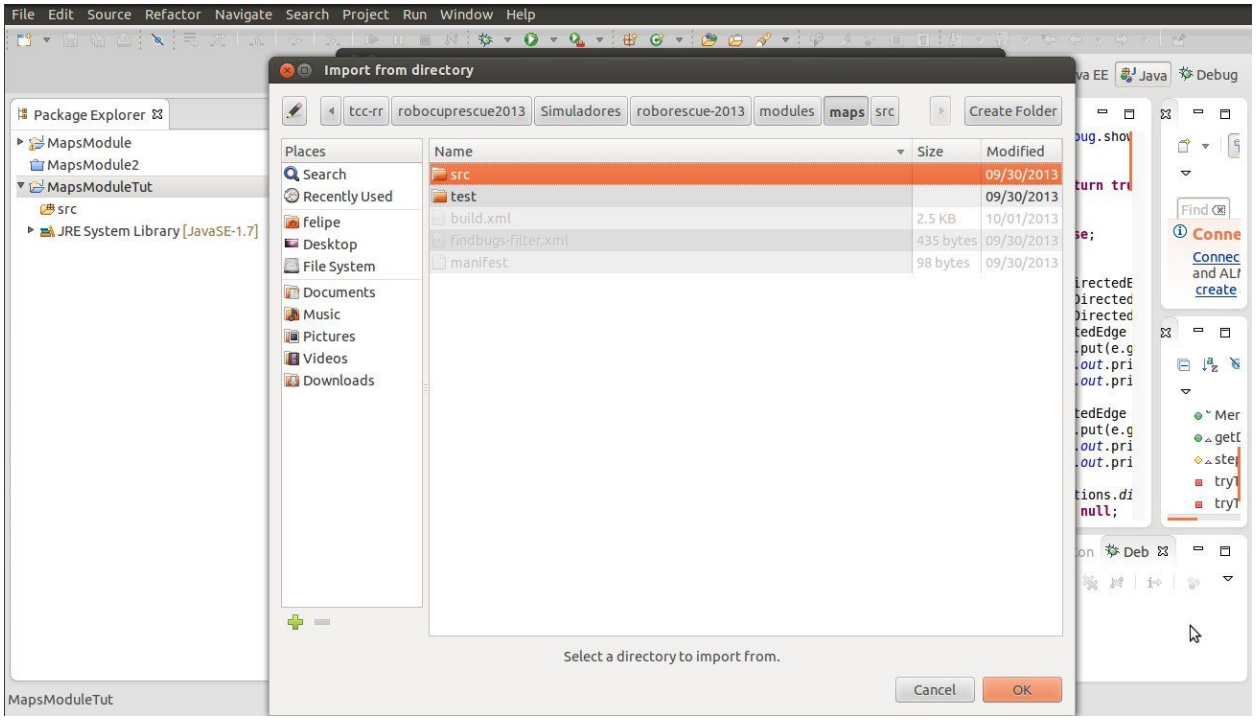

Figure 6 - Importing Library code into Eclipse Workspace

Adding external libraries

The osm2gml library depends on other external libraries provided on the Simulator project folder in two different paths: /jars and /lib. In order to compile the library we will have to add these external libraries to the project. Right click on the project, Build Path -> Configure Build Path -> Libraries -> Add External jars. The following list shows the libraries that need to be added into the project.

- **jars/**
  - rescuecore2.jar
  - gis2.jar
- **lib/**
  - jscience-4.3.jar
  - dom4j.jar
  - uncommons-maths-1.2.jar
  - jaxen-1.1.1.jar
  - jts-1.11.jar
  - common-logging-1.1.1.jar
  - jcommon-1.0.16.jar
  - jfreechart-1.0.13.jar
  - jsi-1.0b2p1.jar
  - log4j-1.2.15.jar
  - resq-fire.jar
  - trove-0.1.8.jar
  - xml-0.0.6.jar

Adapting the OSM map

In order to make the map convertible to the GML format by the LIbrary, some changes will have to be made on the original map, downloaded from the OSM Server, using the JOSM editor. This is because the original maps do not originally mark all the buildings as buildings, which makes them disappear on the conversion process. Also, some roads that are really close to each other may cause conversion problems as well. These are some examples of the changes we will need to make on the original map, in order to make the conversion possible.

Note that the discovery of these problems was purely based on attempts and errors made during the creation of our map for University of Sao Paulo, as some of these issues may not exist, depending on the layout of the original map. The following sections will expose the most recurrent changes we have had to make in order to enable the conversion of our map.

Buildings

Remove Buildings from outermost shapes

In many cases some of the buildings will be located inside territories in the original map, which means that the shapes representing the buildings will be located inside the shape of the territory. In most of the cases, the osm2gml library interprets this set of shapes as only one, the outermost one (i.e the shape of the territory), eliminating all the buildings inside it. In order to avoid that, you will need to remove the outermost shape, allowing the buildings to be processed separately. In order to delete the territory, click in one of its edges, then press “Delete”. The image below illustrates the case of different buildings inside a territorial shape.

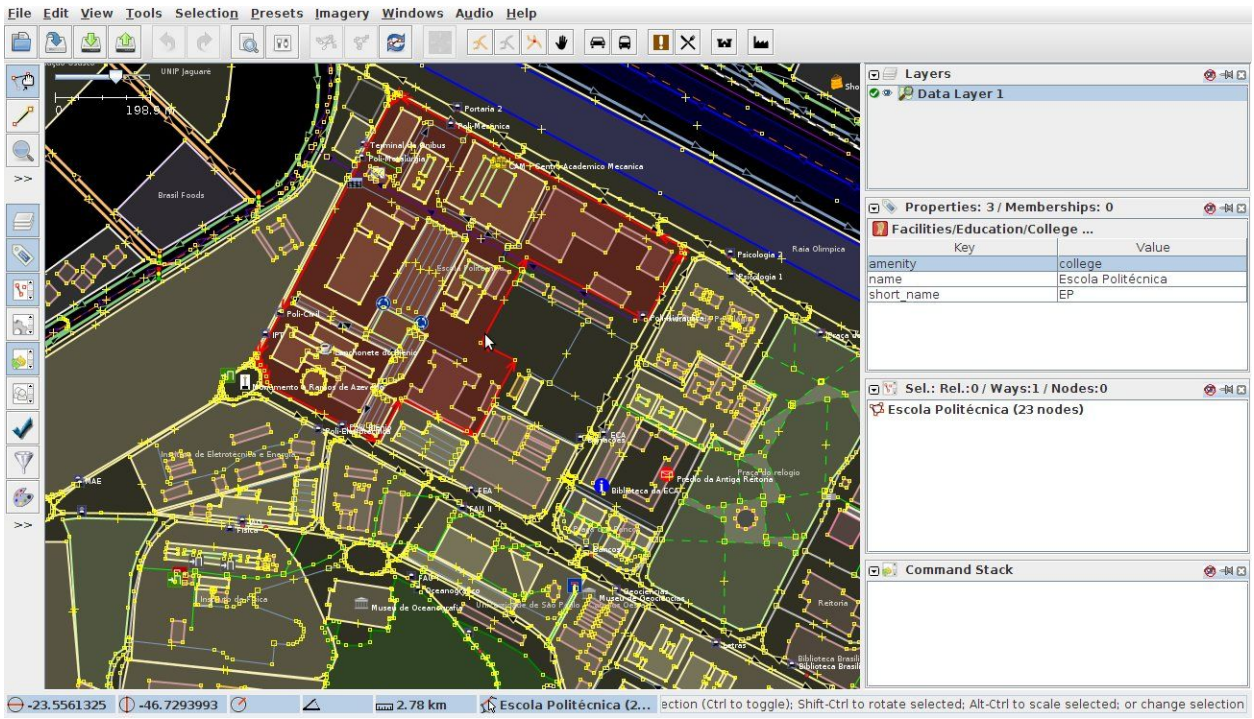

Figure 7 - Outermost shape

Marking Buildings as Building

This is a very important step, because most of the times the shapes corresponding to the buildings in the original OSM map are not marked as buildings, which means that the shape will not be listed as building on the .osm file, and the library won’t be able to identify it as such, eliminating it from the resulting map. In order to configure one shape as a building, select it then go to Presets->Man Made->Man Made->Building, and select one of the types

of building (default is “yes”). The following figure shows the setup of a shape as building.

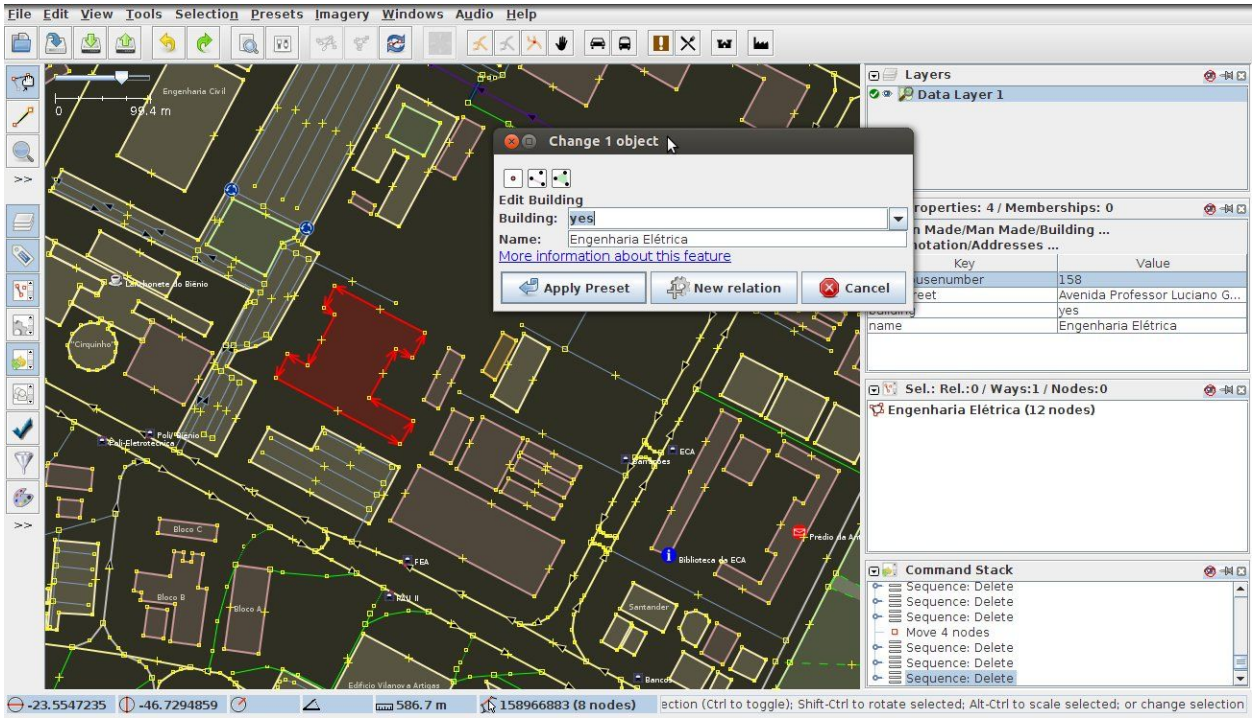

Figure 8 - Building Setup

Separate overlapping buildings

In many occasions, your original map will contain shapes that overlap each other, either two buildings, or one building and a road, or two roads. The converter will process these overlappings (merging or separating) during the conversion process, and sometimes, they may cause it to stop working. So as a safe practice, it is better to separate the overlapping shapes in your map before the conversion. Select one of the shapes and drag and drop it to separate from the other. See the next image as an example of overlapping buildings.

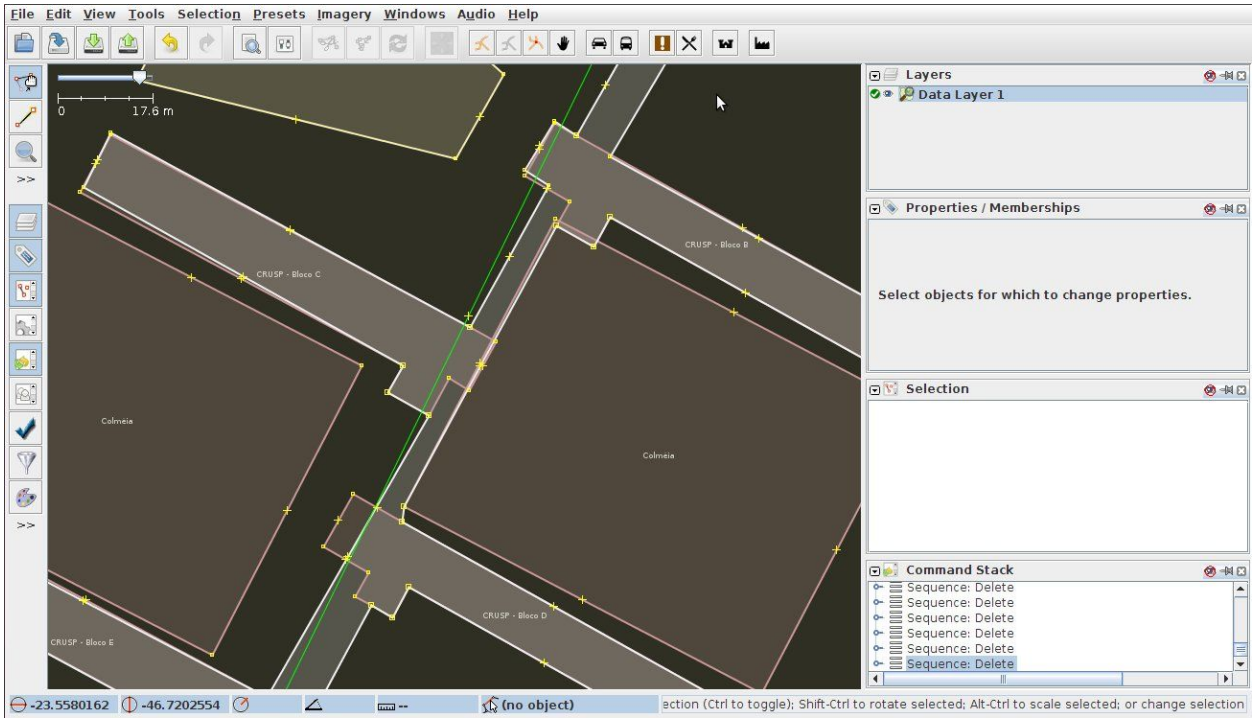

Figure 9 - Overlapping buildings

Roads

Setup roads as both ways

Most of the roads on the original map are set as only one way road, although some of them should be both ways roads, because they connect other roads in different ways (see one example of that on figure 10). In order to set the road as both ways, select it then go to Presets->Highways->Streets and select the desired street type. A dialog will appear, before clicking ok, make sure the check box “Oneway” is not selected. Figure 11 illustrates the dialog for the option of “Unclassified” street.

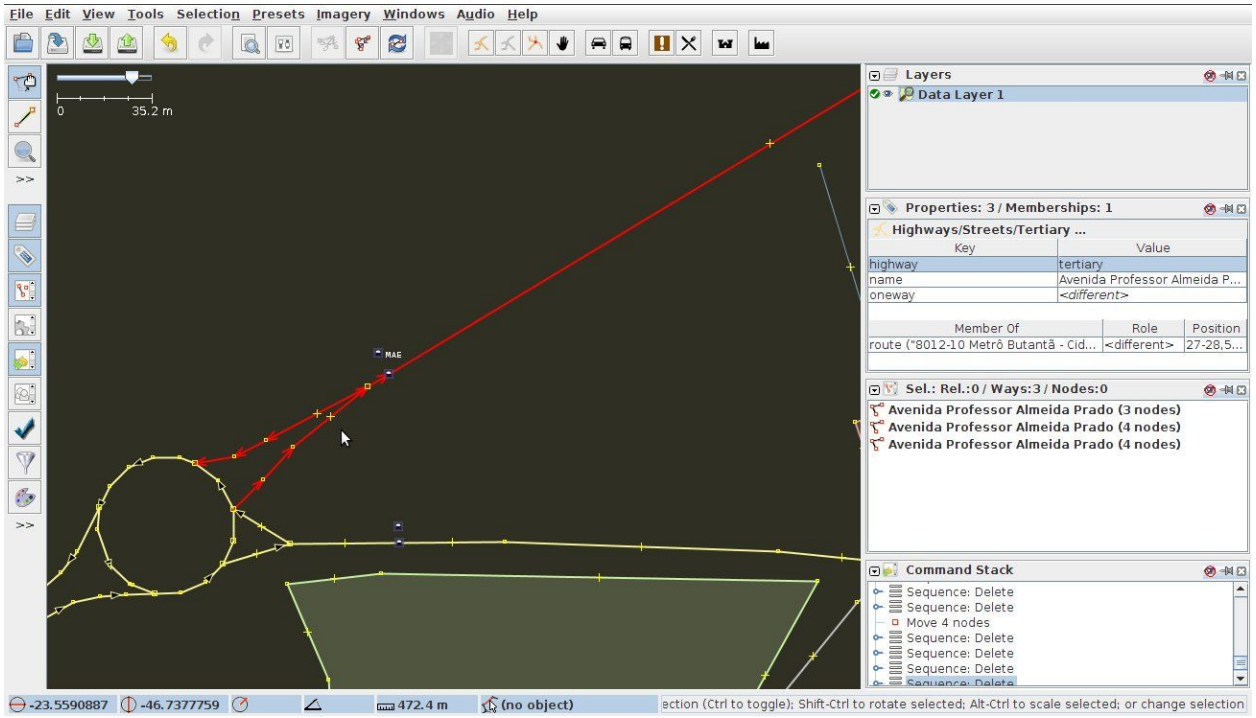

Figure 10 - Both Way Road

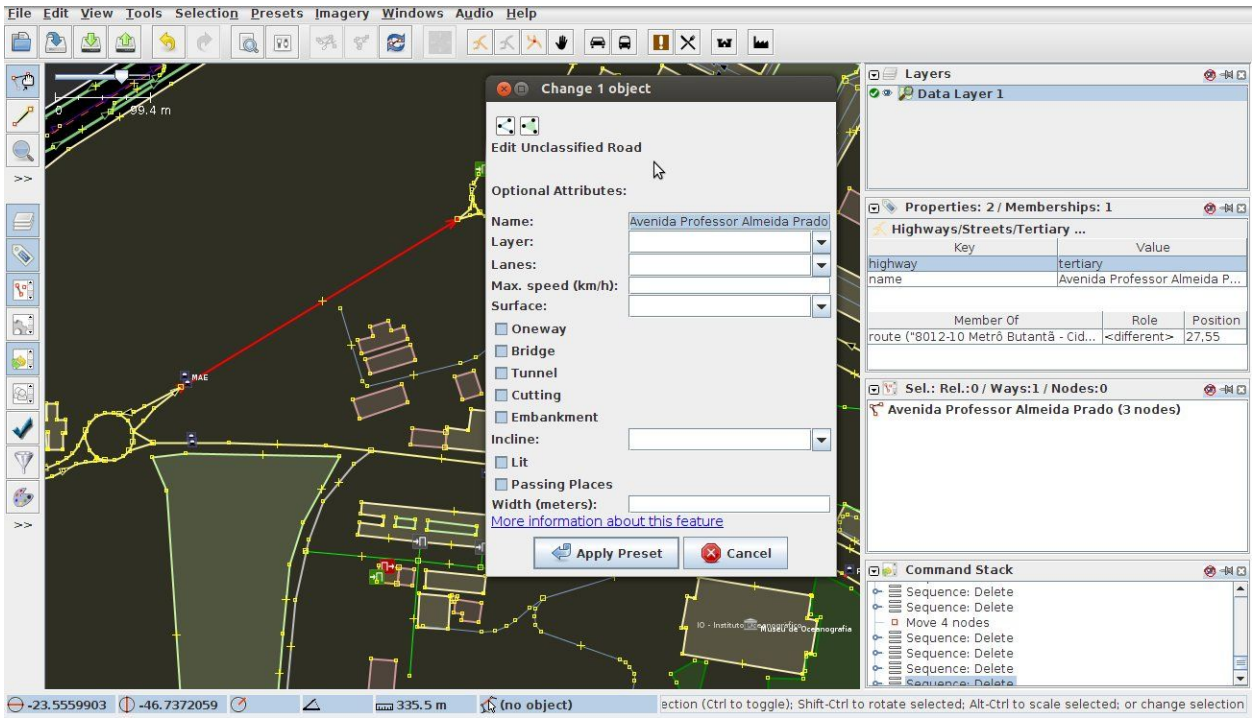

Figure 11 - Setup Dialog

Remove roads from inside buildings

The original OSM maps display some lines inside the buildings, which represent the path one can walk inside them. But sometimes these lines may be interpreted as roads that go inside the building by the osm2gml library, and this can cause the conversion process to stop working. In order to avoid problems like that, you will need to remove these lines from inside the buildings. The image below shows one example of lines displayed inside a building.

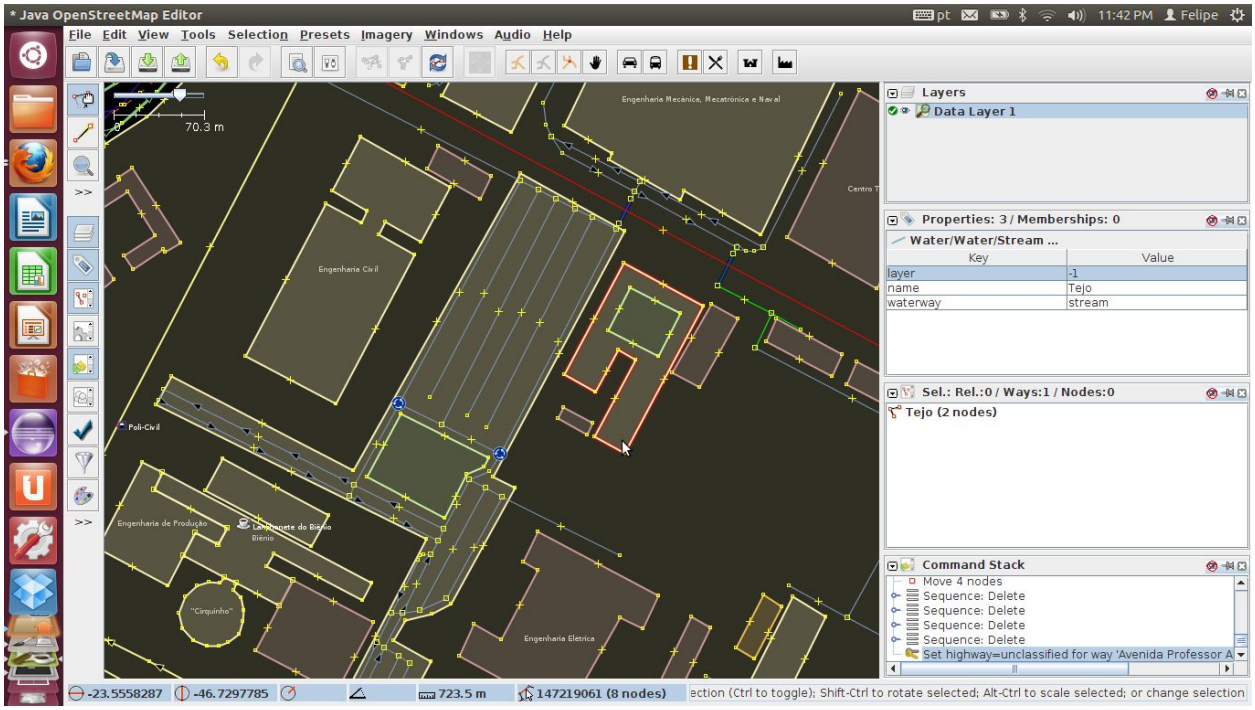

Figure 12 - Line Inside Building

Running the Library

After doing all the necessary changes in the original map, using JOSM, it is now time to run the library in order to convert the map into gml format. In Eclipse, set the run configurations to put the maps.convert.Convert class to launch, using as parameter the path and name of the original .osm file and the destination gml file, in that order. Note that even after following all the procedures above there still may be some invalid pieces in the map that will cause the conversion not to work, and you will have to use JOSM to keep doing the changes described or pinpointing the error location (debugging), therefore this is an iterative process. The following sub-sections will show you how to deal better with errors as well.

The following image illustrates the converter application running, figure 14 shows the resulting map after the application ran successfully:

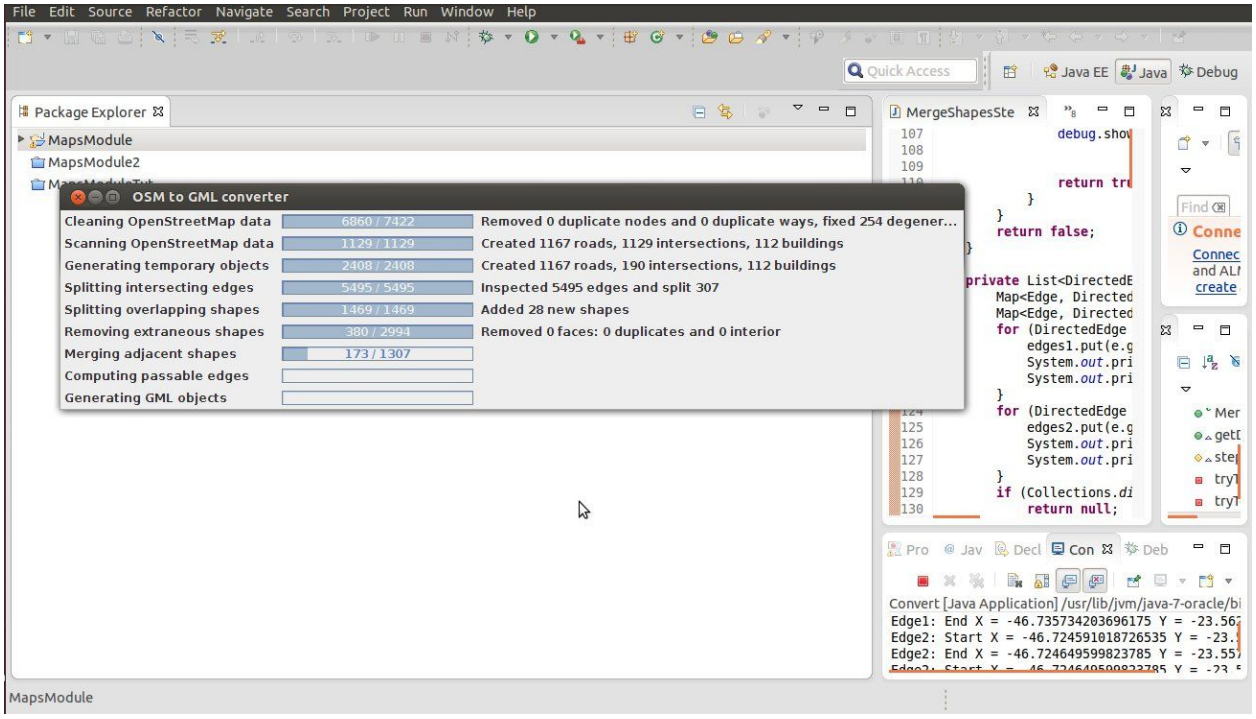

Figure 13 - osm2gml Running

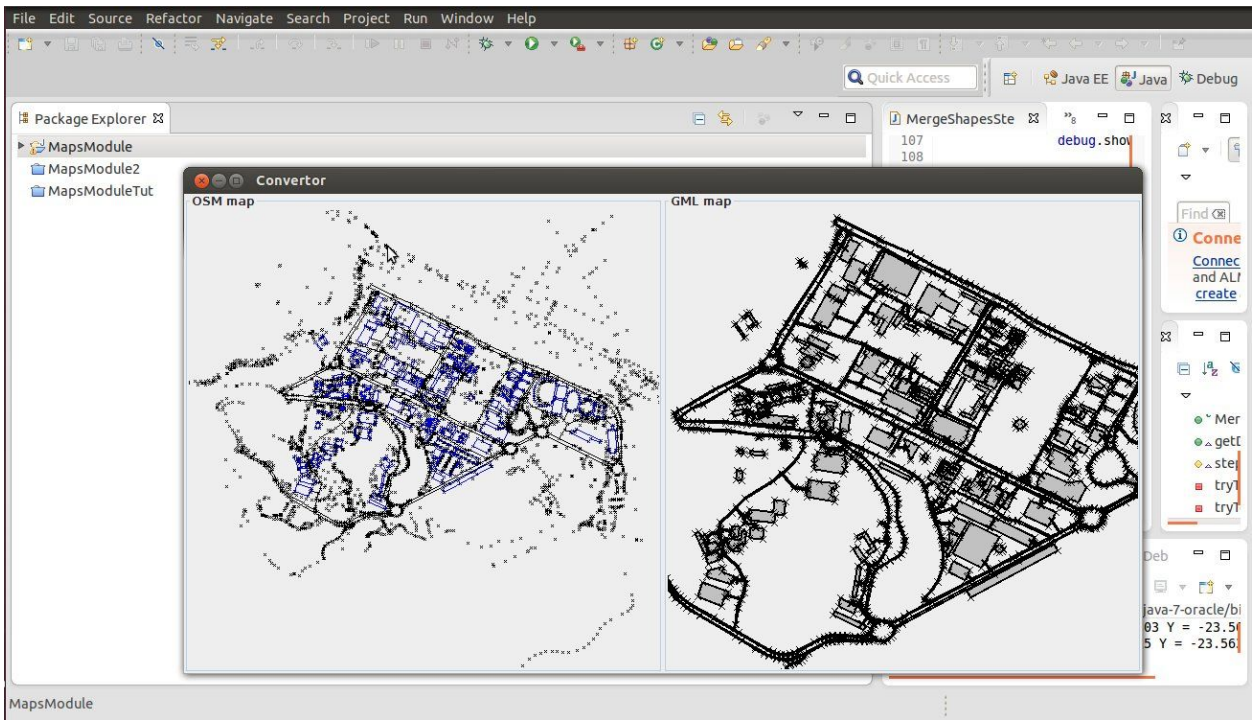

Figure 14 - osm2gml Result

A few tweaks in the library

During the construction of our University of Sao Paulo map, we have realized that many of the buildings and roads were being cut off the resulting map, during the conversion process. Also, we have noticed that most of the parts were being removed in the Step “Remove Shapes Step”, which deletes all objects (roads and buildings) that are duplicates of others, as well as interior faces. To prevent this from happening, we have made a small change on the library code, removing the shape removal part of the code for Buildings and Roads (lines 44,48,51 and 53 commented). Note that normally, changes on the library code are not recommended, but as these changes do not affect the overall behavior of the code, it is a safe strategy to make them. The following figure shows a snapshot of the commented code:

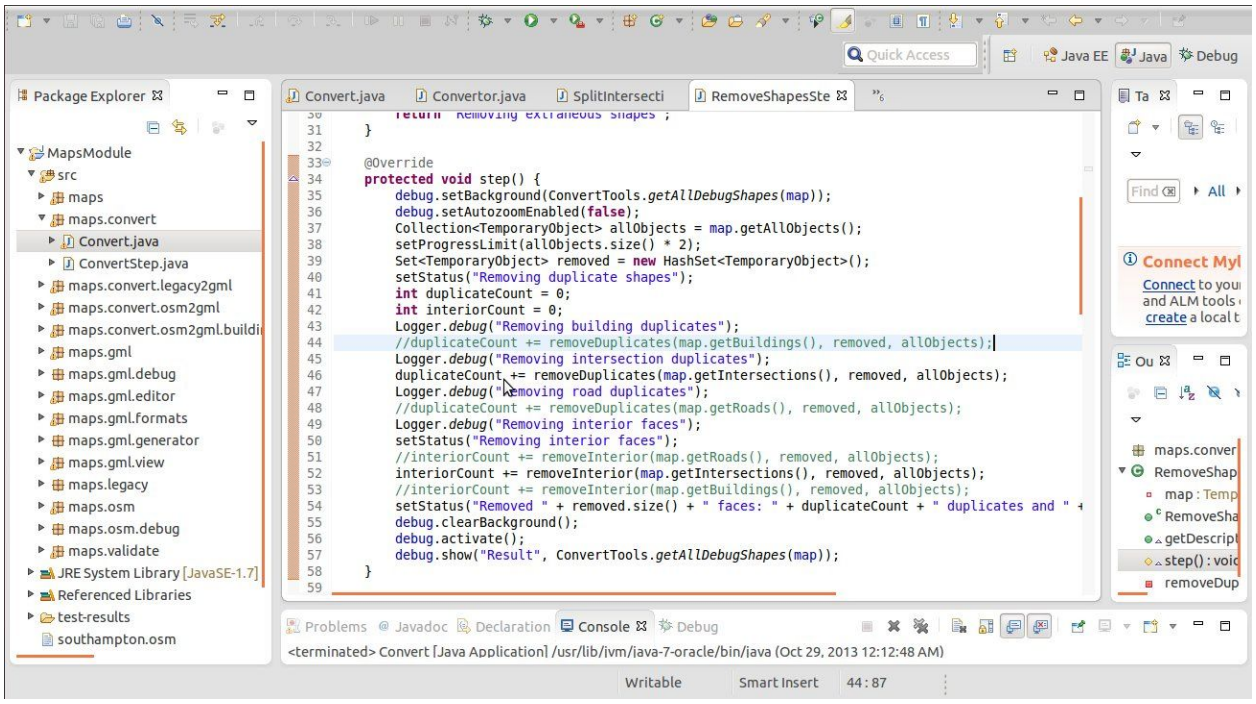

Figure 15 - Library Code commented

Pinpointing an error location

As it was told before, the process of converting the osm file into gml will probably fail for several times before it works out. which means that the conversion process is iterative and based on attempts and errors: after running the library, one can find the error that happened, fix it and run it again. However, sometimes it is really difficult to pinpoint where the error was exactly, in order to do that, we recommend debugging to track which nodes

are being processed at each time to see which one is causing the problem.

6 - Setting up scenario for the map

After creating the map for the simulator, the next step is creating the simulation scenario for the map, which means configuring all the initial simulation setup on your map, such as the initial position of each patrol agent, the locations of the central agents, first buildings on fire, etc. In this section we will show you how to set these configurations for your scenario, using the GML file created last section.

Map directory

Each map/scenario created for the current version of the Simulator is stored inside a specific folder under **/maps/gml/.** Inside the map folder there will be two files: map.gml, created on section 5, as well as scenario.xml with all the scenario configurations that we will be creating on this section. The following image illustrates the directory structure for the map we created in our example case (/maps/gml/USP/).

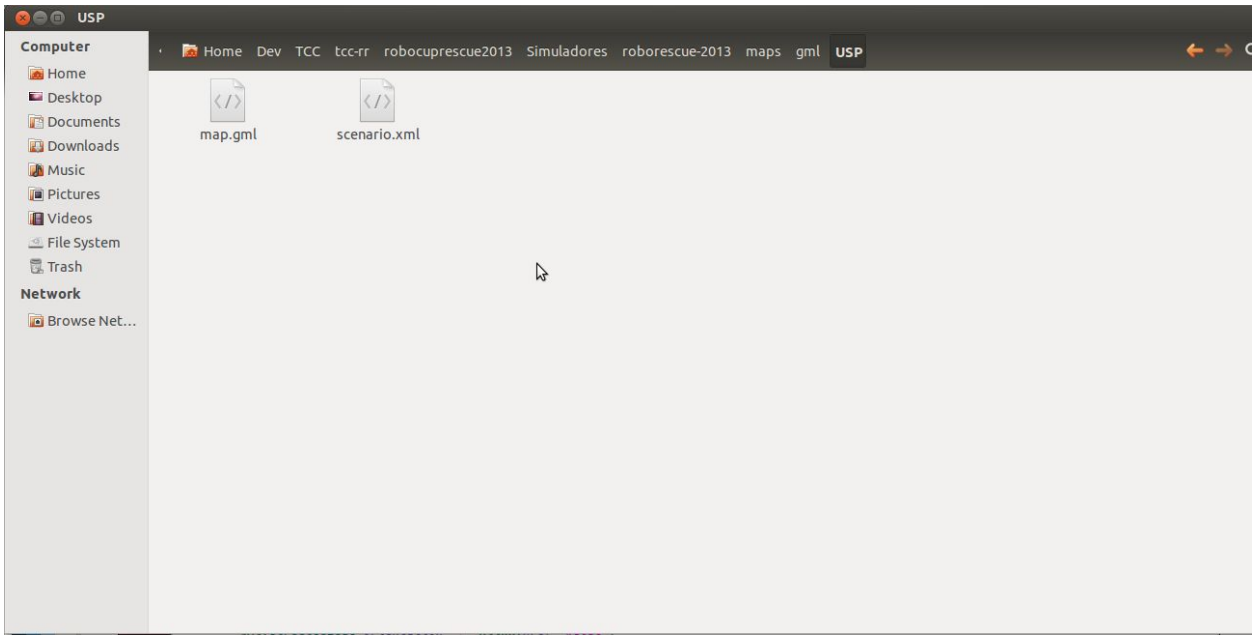

Figure 16 - GML Map Directory Structure

GML file layout

In order to create the scenario, you will need to have an understanding of the layout of the GML file created by the osm2gml library in the previous section. The GML file for the

Robocup Rescue Simulator is basically separated into 4 most important parts: the node list, the edge list, the building list and the road list. The edge list contains description of all the edges, that are composed by the nodes in the node list, while the buildings and roads are defined based on the edges, these last two parts will be really useful on the creation of the scenarios. The following images show a minimized representation of the building and road lists on the GML file generated in the previous section.

Building List

```
<rcr:buildinglist>
  <rcr:building gml:id="8508">
    <gml:Face rcr:floors="1" rcr:buildingcode="0" rcr:importance="1">
      <gml:directedEdge orientation="+" xlink:href="#7724"/>
      <gml:directedEdge orientation="+" xlink:href="#7723"/>
      <gml:directedEdge orientation="+" xlink:href="#3652"/>
      <gml:directedEdge orientation="-" xlink:href="#3654"/>
      <gml:directedEdge orientation="-" xlink:href="#3655"/>
      <gml:directedEdge orientation="+" xlink:href="#3651"/>
      <gml:directedEdge orientation="+" xlink:href="#7720"/>
    </gml:Face>
  </rcr:building>
  <rcr:building gml:id="8509">
    <gml:Face rcr:floors="1" rcr:buildingcode="0" rcr:importance="1">
      <gml:directedEdge orientation="+" xlink:href="#8189"/>
      <gml:directedEdge orientation="+" xlink:href="#8196"/>
      <gml:directedEdge orientation="+" xlink:href="#8195"/>
      <gml:directedEdge orientation="+" xlink:href="#4483"/>
      <gml:directedEdge orientation="-" xlink:href="#4476"/>
      <gml:directedEdge orientation="+" xlink:href="#4479"/>
      <gml:directedEdge orientation="+" xlink:href="#4474"/>
      <gml:directedEdge orientation="+" xlink:href="#8193"/>
      <gml:directedEdge orientation="+" xlink:href="#8192"/>
    </gml:Face>
  </rcr:building>
 ...
  <rcr:building gml:id="8619">
    <gml:Face rcr:floors="1" rcr:buildingcode="0" rcr:importance="1">
      <gml:directedEdge orientation="+" xlink:href="#4280"/>
      <gml:directedEdge orientation="-" xlink:href="#4282"/>
      <gml:directedEdge orientation="+" xlink:href="#4285"/>
      <gml:directedEdge orientation="+" xlink:href="#4284"/>
    </gml:Face>
  </rcr:building>
</rcr:buildinglist>
```

Figure 17 - GML Building List

Road List

```
<rcr:roadlist>
  <rcr:road gml:id="8620">
    <gml:Face>
      <gml:directedEdge orientation="-" xlink:href="#5285" rcr:neighbour="9341"/>
      <gml:directedEdge orientation="+" xlink:href="#6940"/>
      <gml:directedEdge orientation="-" xlink:href="#5655" rcr:neighbour="9515"/>
      <gml:directedEdge orientation="+" xlink:href="#6939"/>
    </gml:Face>
  </rcr:road>
  <rcr:road gml:id="8621">
    <gml:Face>
      <gml:directedEdge orientation="-" xlink:href="#6575" rcr:neighbour="9630"/>
      <gml:directedEdge orientation="+" xlink:href="#7372"/>
      <gml:directedEdge orientation="+" xlink:href="#7371" rcr:neighbour="9958"/>
      <gml:directedEdge orientation="+" xlink:href="#7370"/>
    </gml:Face>
  </rcr:road>
  <rcr:road gml:id="8622">
    <gml:Face>
      <gml:directedEdge orientation="-" xlink:href="#5126" rcr:neighbour="9716"/>
      <gml:directedEdge orientation="+" xlink:href="#5296"/>
      <gml:directedEdge orientation="+" xlink:href="#5295" rcr:neighbour="9827"/>
      <gml:directedEdge orientation="+" xlink:href="#5294"/>
    </gml:Face>
  </rcr:road>
  ...
  <rcr:road gml:id="9984">
    <gml:Face>
      <gml:directedEdge orientation="-" xlink:href="#6149" rcr:neighbour="9351"/>
      <gml:directedEdge orientation="-" xlink:href="#6355" rcr:neighbour="8952"/>
      <gml:directedEdge orientation="-" xlink:href="#4484" rcr:neighbour="8960"/>
    </gml:Face>
  </rcr:road>
</rcr:roadlist>
```

Figure 18 - GML Road List

Creating a scenario

As the GML file, the scenario file is also XML-based, and contains a list of the elements that compose the initial setup for the simulation, including starting fires, refuges, civilians, patrol agents, etc. Each element of the xml file has two attributes. The first of them determines what kind of elements you are creating for the initial configuration (*fire*, *refuge*, *ambulance*, *ambulancecentre*, etc.). The second one is called “Location” and determines where the element will appear in the beginning of the simulation, the value of this attribute is a number that refers to the id of one of the elements in the GML file (either a Building or a Road). The figure below shows a reduced representation of scenario file created for University Of Sao Paulo.

```
<?xml version="1.0" encoding="UTF-8"?>
<scenario:scenario xmlns:scenario="urn:roborescue:map:scenario">
  <scenario:refuge scenario:location="8509"/>
  <scenario:refuge scenario:location="8600"/>
  <scenario:refuge scenario:location="8567"/>
  <scenario:fire scenario:location="8544"/>
  <scenario:fire scenario:location="8601"/>
  <scenario:fire scenario:location="8612"/>
  <scenario:fire scenario:location="8513"/>
  <scenario:fire scenario:location="8577"/>
  <scenario:firestation scenario:location="8598"/>
  <scenario:firestation scenario:location="8589"/>
  <scenario:firestation scenario:location="8590"/>
  <scenario:ambulancecentre scenario:location="8560"/>
  <scenario:ambulancecentre scenario:location="8553"/>
  <scenario:gasstation scenario:location="8533"/>
  <scenario:gasstation scenario:location="8575"/>
  <scenario:civilian scenario:location="8892"/>
  <scenario:civilian scenario:location="9234"/>
  <scenario:civilian scenario:location="9834"/>
  <scenario:civilian scenario:location="8723"/>
  <scenario:civilian scenario:location="9030"/>
  <scenario:firebrigade scenario:location="8999"/>
  <scenario:firebrigade scenario:location="9090"/>
  <scenario:firebrigade scenario:location="8934"/>
  <scenario:firebrigade scenario:location="9645"/>
  <scenario:policeforce scenario:location="9045"/>
  <scenario:policeforce scenario:location="9239"/>
  <scenario:policeforce scenario:location="9139"/>
  <scenario:ambulanceteam scenario:location="9211"/>
  <scenario:ambulanceteam scenario:location="9639"/>
  <scenario:ambulanceteam scenario:location="9268"/>
  <scenario:ambulanceteam scenario:location="9812"/>
  <scenario:hydrant scenario:location="9011"/>
</scenario:scenario>
```

Figure 19 - Scenario Layout

Some types of elements can be located only on Buildings, while the other part can be located only on roads. Note that in the scenario file above the elements are located in the respective type of location (defined in the GML file). The following list shows the types of elements corresponding to Building or Road.

- Building
  - *fire*

    - *refuge*
    - *firestation*
    - *ambulancecentre*
    - *firestation*
    - *gasstation*
- Road
    - *ambulanceteam*
    - *policeforce*
    - *firebrigade*
    - *civilian*
    - *hydrant*

7 - Conclusion

In this tutorial we have shown a couple of basic steps that should be followed in order to create a valid map for the Robocup Rescue Simulator, based on real geographic information. Although the steps described cover basically all the cases and problems we have faced during the creation of our map, there is no official standard procedure for this, as there might be some other issues that were not covered here, in these cases, as was told before, a proper debugging of the conversion library will be really useful. Nevertheless, the previous sections provided a straightforward overview of all the basic concepts required for completing the process, turning this document on a useful reference for any general issue regarding the creation of customized maps and scenarios for the competition Simulator.