# RPL: A Road Planning Language.

FYP Review Report - 2015

B.Sc. Software Development (Hons)

Daniel Desira – 243093(M)

# Table of Contents

# RPL: A Road Planning Language.


## 1. Abstract

Road systems have for ages provided a means of efficient transportation. However, traffic congestion has become more common as growth in human population and economic activity where experienced, bringing several social, environmental, health and economic issues.

RPL's main aim is to allow for the modeling and simulation of road systems, in order to help find ways in which congestion and hence its effects may be minimised. Being a DSL, the solution offers a user-friendly method of analysing road's performance under a specific set of parameters, since the user is alleviated from the graph and queuing theory behind, shifting their focus on constructing the road network.


## 2. Introduction and Background

Over the years, road systems have provided a more efficient means of transportation, but as human populations and economic activity have started to grow, roads have seen a dramatic increase in traffic density[1], rendering the system counter-intuitive. Traffic flow management and proper road planning have since then been sought after by governments and road planning authorities.

Traffic congestion causes driver inconvenience and frustration as well as higher fuel consumption, adding to the driver's frustration as well as air pollution. This higher pollution of air has been linked to fatal diseases and financial burden due to the need for more emphasis on health-care.[2] Moreover, the construction of new roads and maintenance of existing ones is expensive in terms of both public funds and land use, hence the most worthwhile network changes should be determined in order to cut on these costs. More road accidents and fatalities happen due to traffic congestion, making people feel unsafe on the road. The longer commutes resulting from congestion are also said to contribute to obesity.[3]

## 3. Aims and Objectives

The project's main objective is to enable better planning of maintenance to road systems in order to improve their capability in handling the increasing demand, due to more accurate analysis on the effects that some change would have on traffic flow across the road network.

## 4. Design

The proposed solution is a language that models road systems and allows for a simulation of what happens to an existing road network with a set of parameters and how the system would behave if a modification is performed.

The road system is modeled by constructing a graph, while simulation is handled through the use of M/M/1[1] queues. Graph creation and manipulation is in turn handled by the JGraphT library and the details of simulation are taken care of by PDQ[2], a tool focused on computer-based performance analysis. The diagram that follows demonstrates the connection between the various components from a top-level perspective:

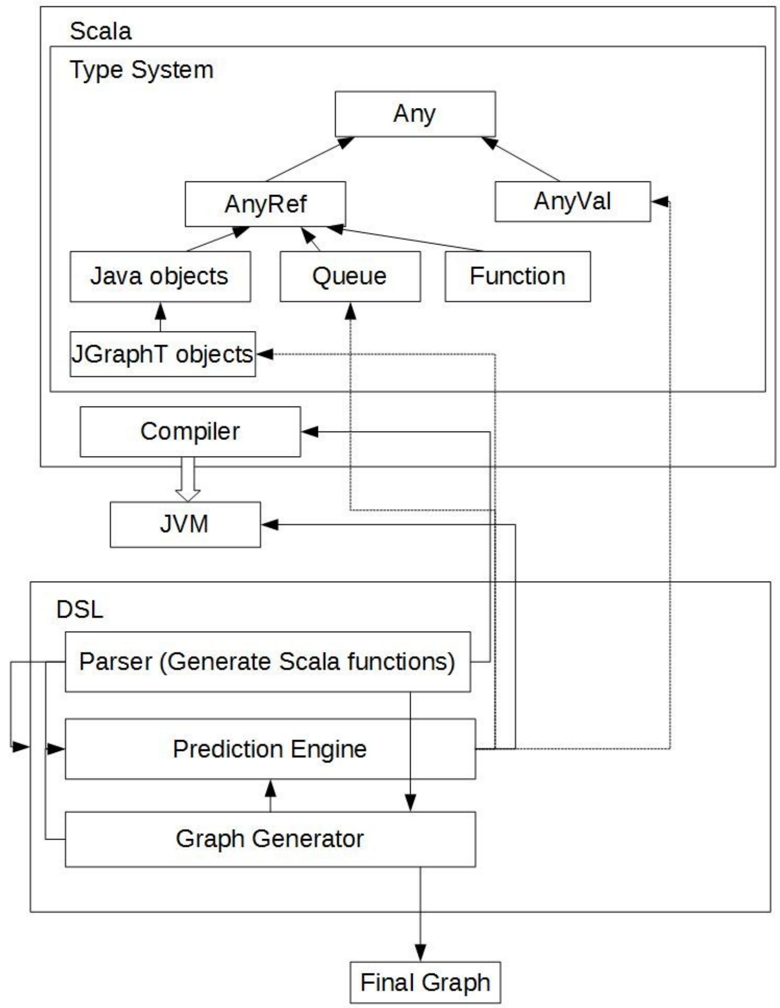


**Figure 1: Solution top-level block diagram**

The proposed language consists of several constructs that help represent roads, their connections and three of the most common road elements: roundabouts, T-junctions and crossroads. It also assumes a two-way road

1 a queuing system having a single server, where arrivals have a Poisson disribution and job service times have an exponential distribution
2 Pretty Damn Quick

system and allows for the definition of various parameters such as the number of lanes in a street. Below is the language's definition in EBNF style:

```
model = constructNetwork, definitions,
runSimulation, [modifications];


constructNetwork = "construct",
"network", identifier, "(";


definitions = createRoad, { attachRoad |
crossroad | roundabout | createRoad },
")";
createRoad = "create", "primary",
"road", identifier, "with", "length",
floatingPointNumber, "left", "has",
"vehicles", (wholeNumber | "?"),
"arrival", "rate", (floatingPointNumber
| "?"), "right", "has", "vehicles",
(wholeNumber | "?"), "arrival", "rate",
(floatingPointNumber | "?"), ["lanes",
wholeNumber];
attachRoad = "attach", ("primary" |
"secondary"), "road", identifier,
"with", "length", floatingPointNumber,
["to", identifier], "at",
floatingPointNumber, "left", "has",
"vehicles", (wholeNumber | "?"),
"arrival", "rate", (floatingPointNumber
| "?"), "right", "has", "vehicles",
(wholeNumber | "?"), "arrival", "rate",
(floatingPointNumber | "?"), ["lanes",
wholeNumber];
crossroad = "crossroad", identifier,
"at", floatingPointNumber, "with",
identifier, "at", floatingPointNumber;
roundabout = "roundabout", "on",
identifier, "at", floatingPointNumber,
"exit", "rate", floatingPointNumber;


blockRoad = "block", identifier;

change = "change", identifier, "lanes",
wholeNumber;


runSimulation = "run", "simulation",
"for", "minutes", floatingPointNumber;


modifications = { blockRoad | change |
roundabout }+, "rerun";
```

The proposed road elements are representable through graphs and the following graph translation scheme has been chosen:

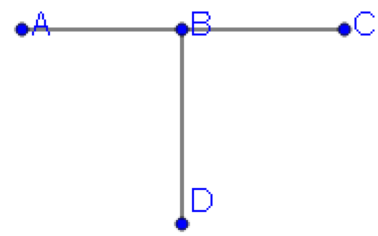


**Figure 2: T-junction**

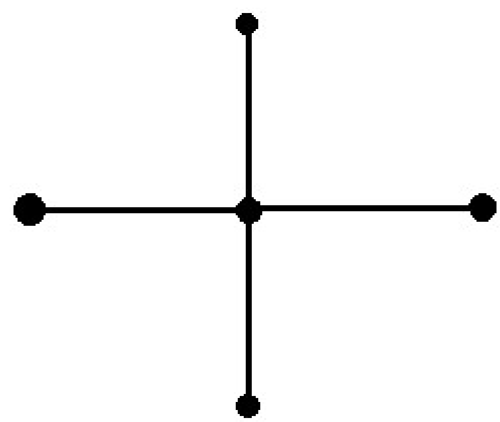

**Figure 3: Crossroads**

**Figure 4: Minimal roundabout**

The core functionality is called by the language so as to model the intended road system effectively. In RPL's case, the functionality is implemented by calling methods from JGraphT and PDQ, as illustrated by the following table:

<table>
<tr><th>Statement/Construct</th><th>Procedure Call</th></tr>
<tr><td><code>construct Network <network_name></code></td><td><code>network = new DirectedGraph()</code></td></tr>
<tr><td><code>create|attach primary|secondary road <road_name></code></td><td><code>v1 = network.addVertex()</code><br><code>v2 = network.addVertex()</code><br><code>network.addEdge(v1, v2)</code></td></tr>
<tr><td><code>block <road_name></code></td><td><code>network.removeEdge(v1, v2)</code><br>OR<br><code>network.removeEdge(e)</code></td></tr>
<tr><td><code>add semaphore| traffic light| zebra crossing|...... at <road_name> <position(integer)></code></td><td><code>network.addVertex(...)</code><br>Divide the edge in two.</td></tr>
<tr><td><code>attach secondary road <road_name> two way</code></td><td>Create 2 M/M/1 queues using PDQ</td></tr>
<tr><td><code>construct network <network_name> (......) [construct network <network_namex> (.......)]* simulate *| <network_name>[, <network_namex>]*</code></td><td>PDQ-based model should take care of the simulation process, after the graph is built:<br><br><code>pdq = new PDQ</code><br><code>pdq.Init(modelName)</code><br><code>pdq.CreateNode(...)</code><br><code>pdq.CreateClosed(...)</code><br><code>pdq.Solve(....)</code><br><code>pdq.Report()</code></td></tr>
<tr><td><code>Simulate .... given <amount> vehicles in <road_name> [, <amount> vehicles in <road_name>]*</code></td><td><code>pdq.CreateClosed(...)</code></td></tr>
</table>

# 5. Implementation

RPL is written in Scala because of the language's expressiveness in that both Object-Oriented and functional styles may be used.

The language is implemented through the use of Scala Parser Combinators. Besides being usable through SBT, this library provides the advantage of enabling developers to define parsers functionally, such that the implementation is similar to the model's EBNF representation.[4]

JGraphT allows for easier manipulation of graphs, simplifying the solution's graph generation logic.

On the other hand, PDQ is library that allows for simulation based on M/M/1 queues. RPL uses the closed-circuit queuing system due to traffic flow's finite nature. It is worth-mentioning that PDQ works similar to a DSL whereby one is expected to define a model describing the queue(s) to be initialised along with their servers. This renders PDQ's operation imperative and hence calls to its functions may be easily encapsulated in a few methods.[5]

The class diagram below displays a simple overview of the relationships between classes and objects and some of the library code called by RPL:

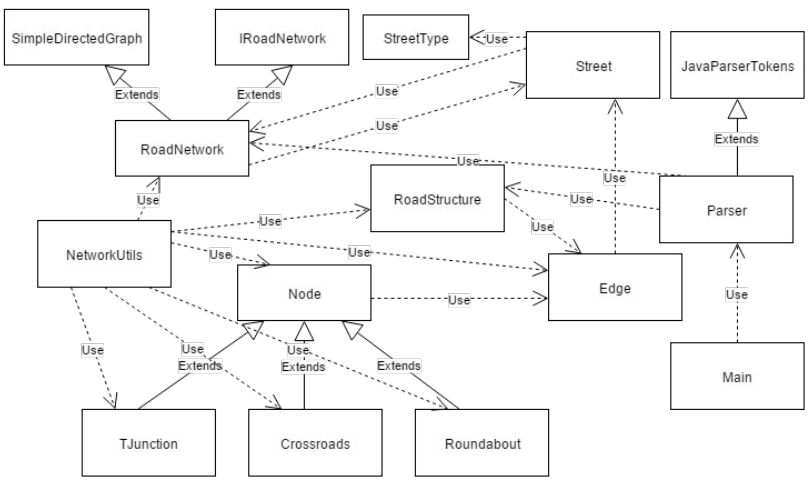


**Figure 5: Solution class diagram**

# 6. Results and Evaluation

RPL is capable of representing road systems in a rather accurate manner, allowing for three of the most commonly occurring road elements so as to make the process more realistic.

In the model that follows, two roads are attached at the ends of the main street, while one is attached in between the ends forming a T-junction:

```
construct network zabbar_primaries (
      create primary road sanctuary_road
with length 700 left has vehicles 10
arrival rate 9.2 right has vehicles 2
arrival rate 0
      attach primary road hompesch_road
with length 800 at 700 left has vehicles
30 arrival rate 22.5 right has vehicles
3 arrival rate 0 lanes 1
      attach primary road
marsascala_old_road with length 1600 at
200 left has vehicles 8 arrival rate 6.9
right has vehicles 4 arrival rate 0
      attach primary road barun_road
   with length 300 at 0 left has
   vehicles 7 arrival rate 7.6 right
   has vehicles 2 arrival rate 0
   )

   run simulation for minutes 1
```

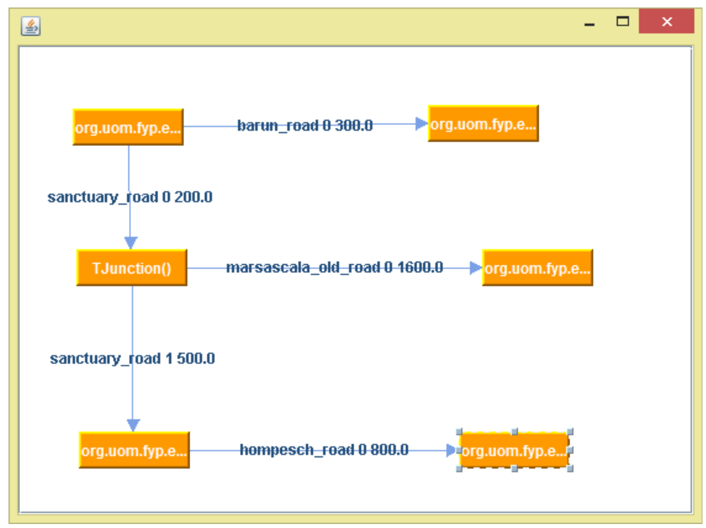


**Figure 6: Road attachment test**

The following example shows the implementation of a crossroads together with the resulting graph:

```
construct network crossroad_test (
      create primary road triq_farfett
with length 200 left has vehicles 0
arrival rate 0 right has vehicles 0
arrival rate 0
      create primary road triq_qattus
with length 230 left has vehicles 0
```

```
arrival rate 0 right has vehicles 0
arrival rate 0
      crossroad triq_farfett at 90 with
triq_qattus at 117.5
)
run simulation for minutes 1
```

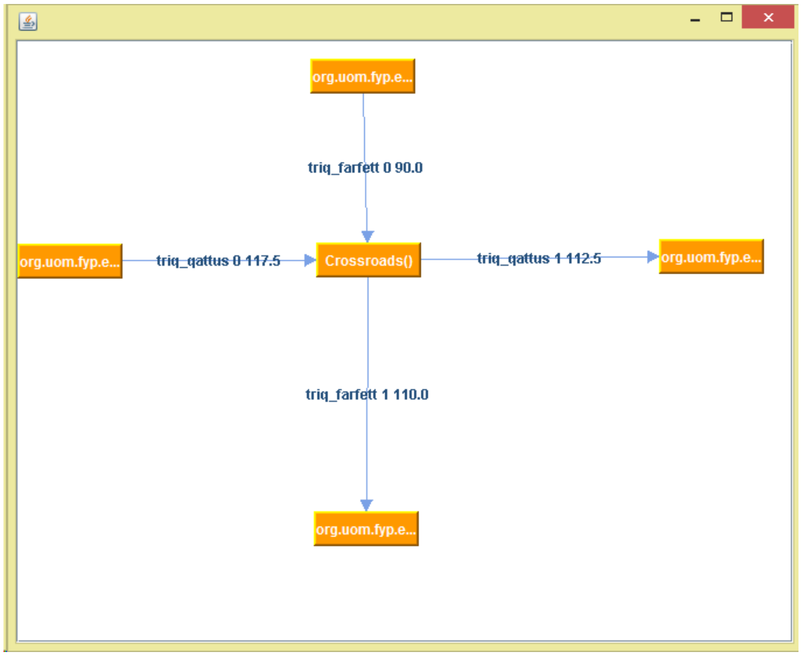


**Figure 7: Crossroads test**

Moreover, one is also somehow allowed to define roundabouts in their models, as in the RPL program that follows:

```
construct network roundabout_test (
      create primary road abc with
length 300 left has vehicles 0 arrival
rate 0 right has vehicles 0 arrival rate
0
      roundabout on abc at 150 exit rate
0
)
run simulation for minutes 1
```

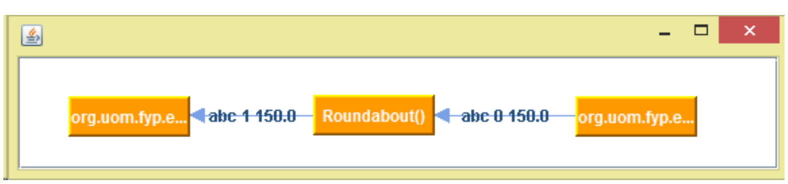


**Figure 8: Roundabout test**

By allowing the user to compare the performance of an existing road system with that of the resulting system after a set of changes, the proposed solution helps to improve the road construction and traffic flow management process and alleviate the pain associated to traffic congestion.

Moreover, the system allows for the omission of certain parameters in order for those parameters to be input on demand, rendering the solution more adaptable to changes in traffic flow reflected by changes in human population and economic activity.

## 7. Conclusions and Future Work

The solution proves that road systems may be modeled using a DSL, given that the appropriate algorithms and data structures are implemented behind the scenes.

The solution may be improved by supporting more road elements such as zebra crossing, traffic lights and others, re-factoring the project to act as a REST API[3] and

3 Web Service based on the REST architecture

a web client that displays a more graphical representation of the various statistics as well as helping to better identify bottlenecks.

## 8. Bibliography

[1] A. Dawns. “Traffic: Why It's Getting Worse, What Government Can Do.“ Internet: http://www.brookings.edu/research/papers/2004/01/01transportation-downs, Jan, 2004 [Aug. 30, 2015].

[2] K. Dalli. “Polution now killing 230 people every year.” Internet: http://www.timesofmalta.com/articles/view/20150429/local/pollution-now-killing-230-people-every-year.565997, Apr. 29, 2015 [Aug. 15, 2015].

[3] Dr. J. P. Rodrigue. (2013). *The Geography of Transport Systems.* (3rd edition). [On-line]. “Urban Transport Challenges.” Available: https://people.hofstra.edu/geotrans/eng/ch6en/conc6en/ch6c4en.html [May 3, 2015].

[4] D. Ghosh. “External DSLs made easy with Scala Parser Combinators”. Internet: http://debasishg.blogspot.com/2008/04/external-dsls-made-easy-with-scala.html, Apr. 14, 2008 [May 10, 2015]

[5] “PDQ: Pretty Damn Quick.” Internet: http://www.perfdynamics.com/Tools/PDQ.html [Aug. 21, 2015]