Literate Programming Report

Kevin Carmona-Murphy

April 5, 2016

Abstract

roadnet is a Ruby based compiler for converting an XML description of a road network into an HTML/SVG visual representation. This document outlines the methodology, languages and frameworks used, as well as implenetation and testing steps.

1 Summary

roadnet is the name given to a tool which allows for the generation of simple graphical road networks using a specially developed road topology language. This tool converts an XML file containing the hierarchy of the road network into an HTML page displaying the network from a bird's-eye viewpoint, using SVG drawings. Three different road types are supported, including a two-lane road, a local street, and a four lane avenue. Road length, intersection radius, and intersecting angle offsets are the parameters that are specifiable, in addition to the structure of the network, via the XML file.

1.1 Problem Inspiration

The inspiration for the tool stems from a pass-time I once cherished as a child. I would often spend hours sketching small neighbourhoods onto a large piece of drafting paper. Attention to detail was especially evident in how I drew the roads; from a bird's eye perspective, lane widths were meticulously calculated and enforced with a ruler through the entirety of my drawings.

Attempting to bring rekindle that hobby and bring it into the digital world is the motivation for such a tool. While of no practical purpose for transportation engineers, it offers an academic and practical insight into the fundamentals of compiler design with recursion, using a modern high-level language.

2 Languges & Tools

The compiler wtransforms an XML file into an HTML file with embedded SVG graphics. XML was chosen as the source because it allows one the freedom to

write nested hierarchies with tags that are describe attributes. This is important given that a road topology is a network of elements that are often repeated and are always connected to one another. An XML file assures that each element has a parent, just like each segment of road generally intersects with another.

SVG within HTML is used as the output representation because it is a very widely available and performant markup language which is compatible across platforms. Using SVG, **roadnet** is able to easily create paths that represent lane markings, road edges, and intersections.

The program is written in Ruby in order to take advantage of the excellent Nokogiri gem which makes the task of parsing elements in XML and HTML files a breeze. In the implementation, Nokogiri parses the XML file recursively, drawing the appropriate paths in SVG as it descends the parse tree.

3 Resources

Various resources were consulted over the course of the development of the tool.

Resources consulted over the course of the project:

- **General inspiration:** http://programmers.stackexchange.com/questions/165543/how-to-write-a-very-basic-compiler
- **Nokogiri docs:** https://github.com/sparklemotion/nokogiri
- **Nokogiri cheatsheet:** https://github.com/sparklemotion/nokogiri/wiki/Cheat-sheet
- **SVG primer:** http://www.w3schools.com/svg/svg_examples.asp
- **LaTeX information:** https://en.wikibooks.org/wiki/LaTeX
- **NoWeb literature:** http://www.literateprogramming.com/noweb_hacker.pdf

4 Implementation

4.1 Getting Started

Running **roadnet** is very simple. In the directory where parser.rb is found, simply execute the following command:

```
$ ruby parser.rb -f rd.xml
```

The -f command line option is necessary - it specifies the road topology xml file. An -o flag is optional, it specifies the output file. The default is diagram.html

4.2 The XML Source

The XML file must have exactly one ⟨network⟩root tag, followed by any number of nested structures comprising of ⟨road⟩, ⟨avenue⟩, or ⟨street⟩tags. Including other tags will raise an error. A parent tag of child elements suggests that the child elements are connected via an intersection to the end of the parent tag.

There are four specifiable attributes. The *intersection-offset* and *intersection-radius* are intersection level attributes, and specify properties pertaining to an intersection, while the *length* attribute and *angle-offset* specify properties of a network link (be it a road, street, or avenue). *intersection-offset* is specified in degrees, and rotates the intersection and all jutting links by x degrees. *intersection-radius* specified in pixels, dicates the radius an intersection. *angle-offset* rotates the specific link on which it is called x degrees clockwise. *length* is self explanatory, its dimensions are in pixels.

Here is an example XML file illustrating the abovementioned properties:

```
<?xml version="1.0" encoding="UTF-8"?>

<network intersection-radius="100">
        <avenue>
        </avenue>
        <road intersection-offset="180"
        angle-offset="-20" intersection-radius="20">
                <road>
                </road>
                <street>
                </street>
        </road>
        <road>
        </road>
        <road>
                <street>
                </street>
                <street length="400" >
                        <road>
                        </road>
                        <road>
                        </road>
                        </street>
                <street>
                </street>
        </road>
        <street>
        </street>
</network>
```

Due to the recursive tree-like structure of the XML file, no cycles are permitted in the road network. That is, a nested link cannot connect with with another beyond its immediate siblings. In other words, the set forms a minimum spanning tree, and is divergent, rather than convergent.

4.3 Ruby Program

There are three dependencies that must be installed using a tool such as **bundler** before running the program. These dependencies include **nokogiri**, **optparse**, and **solid_assert**.

The below code executes the option parser which looks for the command line arguments specifying the input xml file, and an optional output html file. If the input file is missing, or there are weird command line arguments, and error is thrown and the program is aborted.

```
#!/usr/bin/env ruby

require 'rubygems'
require 'bundler/setup'

require 'nokogiri'
require 'optparse'
require 'solid_assert'

SolidAssert.enable_assertions

options = {}

optparse = OptionParser.new do |opts|
        opts.banner = "Usage: parser.rb -f <xml filename> [ -o <output filename>
        opts.on('-f', '-x', '--xml input-filename', 'XML file name') { |v| optio
        opts.on('-o', '-h', '--html output-filename', 'HTML file name') { |v| op
end

begin
        optparse.parse!
        mandatory = [:filename]
        missing = mandatory.select{ |param| options[param].nil? }
        unless missing.empty?
    puts "Missing options: #{missing.join(', ')}"
    puts optparse
    exit
  end
```

```
rescue OptionParser::InvalidOption, OptionParser::MissingArgument
        puts $!.to_s # Friendly output when parsing fails
  puts optparse
  exit
end
```