Software for Interactive Metro Style Map
Visualisation of Crew Scheduling Data

Mantas Gudynas
Computer Science (Ind)
(2009/2010)

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

Signature of student: _________________________
Summary

This project is dealing with the problem faced by users of TrainTRACS crew scheduling software by Tracsis plc. Using the software involves the extensive configuration stage. The aim of this project was to investigate the process of setting up route knowledge table and attempt to replace current practice of use matrix based interface with a metro style map based visualisation of geographical data. Software system based on Model-View-ViewModel architectural design pattern encompassing graphical user interface and hill climbing optimisation algorithm logic. Software architecture was modelled and implemented using Windows Presentation Foundation UI framework. Evaluation of software usability was performed in order to compare the two interfaces.
Acknowledgements

I would like to thank my supervisor Dr Raymond Kwan for his guidance throughout this project and a number of brainstorming discussions.

I would also like to say big thanks to Dr Andrew Schwarz of Tracsis plc. who put me on the right track for the technology.

Finally I would like to thank the Tracsis team and John McArthur in particular for all the input from the company.
## Contents

1 Introduction .......................................................... 1
1.1 Outline ................................................................. 1
1.2 Aim ................................................................. 1
1.3 Objectives ............................................................ 2
1.4 Requirements ......................................................... 2
  1.4.1 Minimum Requirements .......................................... 2
  1.4.2 Possible Extensions ............................................... 3
1.5 Deliverables .......................................................... 3
1.6 Relevance to Degree .................................................. 3
1.7 Project Schedule ....................................................... 3

2 Background .............................................................. 4
2.1 Metro Maps ............................................................. 4
  2.1.1 Overview .......................................................... 4
  2.1.2 Characteristics ..................................................... 4
    2.1.2.1 Shape generalisation ........................................ 5
    2.1.2.2 Angle generalisation ........................................ 5
    2.1.2.3 Uniform spacing of locations ................................ 6
  2.1.3 Brief History ...................................................... 7
2.1.4 Automated Layout .................................................. 9
  2.1.4.1 Multicriteria Hill Climbing ................................... 9
  2.1.4.2 Force-Based Algorithms ...................................... 9
  2.1.4.3 Other Heuristics ............................................... 10
  2.1.4.4 Mixed Integer Programming ................................... 10
  2.1.4.5 Summary ...................................................... 10

3 Technology ................................................................. 11
3.1 Windows Presentation Foundation .................................. 11
  3.1.1 Overview ........................................................ 11
  3.1.2 XAML ............................................................. 12
### 3.1.3 Visual and Logical Trees

4

### 3.1.4 Styles

5

### 3.1.5 Templates

6

### 3.1.6 Data Binding

7

### 3.1.7 Commands

8

### 3.2 Model-View-ViewModel Architectural Design Pattern

9

### 3.3 Summary

10

### 4 Algorithm

11

#### 4.1 Overview

12

#### 4.2 Preprocessing

13

#### 4.3 Aesthetic Criteria

14

##### 4.3.1 Angular Resolution

15

##### 4.3.2 Edge Length Criterion

16

##### 4.3.3 Line Straightness

17

##### 4.3.4 Octilinearity

18

##### 4.3.5 Balanced Edge Length

19

#### 4.4 Node Movement

20

##### 4.4.1 Node Movement Procedure

21

#### 4.5 Iterations

22

### 5 Implementation

23

#### 5.1 Overview

24

#### 5.2 Model

25

#### 5.3 View

26

##### 5.3.1 Map Container

27

##### 5.3.2 Map Objects

28

#### 5.4 ViewModel

29

##### 5.4.0.1 ViewModel of Map

30

##### 5.4.0.2 ViewModel of Location

31

##### 5.4.0.3 ViewModel of Leg

32

#### 5.5 Review of Software Flexibility

33

##### 5.5.1 Replacement of Data Templates for Visual Elements

34

##### 5.5.2 Introduction of a New Command

35

### 6 Evaluation

36

#### 6.1 Evaluation of the software usability

37

##### 6.1.1 Evaluation Framework

38

##### 6.1.2 Evaluation Results

39
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2 Evaluation of the Algorithm Performance</td>
<td>41</td>
</tr>
<tr>
<td>6.2.1 Fictional Sodor network</td>
<td>41</td>
</tr>
<tr>
<td>6.2.2 Virgin West Coast network</td>
<td>42</td>
</tr>
<tr>
<td>6.2.3 Conclusion</td>
<td>44</td>
</tr>
<tr>
<td><strong>Bibliography</strong></td>
<td>45</td>
</tr>
<tr>
<td>A Personal Reflection</td>
<td>47</td>
</tr>
<tr>
<td>B Letters of Support</td>
<td>48</td>
</tr>
<tr>
<td>C Gantt Charts</td>
<td>51</td>
</tr>
<tr>
<td>D Empirical Evaluation Test</td>
<td>53</td>
</tr>
<tr>
<td>E Custom Implementation of UndoableCommand</td>
<td>54</td>
</tr>
<tr>
<td>F Screenshot of the Software</td>
<td>56</td>
</tr>
<tr>
<td>G Empirical Evaluation Results</td>
<td>57</td>
</tr>
<tr>
<td>H Algorithm Solutions</td>
<td>59</td>
</tr>
<tr>
<td>H.1 Sodor Network</td>
<td>59</td>
</tr>
<tr>
<td>H.2 Virgin West Coast Network</td>
<td>61</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Segment of map before (above) and after applying shape generalisation</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Segment of map before (above) and after applying angle generalisation</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>Segment of map before (above) and after applying uniform spacing</td>
<td>6</td>
</tr>
<tr>
<td>2.4</td>
<td>Map of London Metropolitan Railway Network (1986)</td>
<td>8</td>
</tr>
<tr>
<td>2.5</td>
<td>Metro Map for London Underground by Harry Beck (1933)</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>Declarative definition of a user interface in XAML</td>
<td>12</td>
</tr>
<tr>
<td>3.2</td>
<td>Imperative definition of a user interface in C#</td>
<td>13</td>
</tr>
<tr>
<td>3.3</td>
<td>XAML code for the example of element trees</td>
<td>14</td>
</tr>
<tr>
<td>3.4</td>
<td>Example of a logical tree</td>
<td>14</td>
</tr>
<tr>
<td>3.5</td>
<td>Example of a visual tree</td>
<td>15</td>
</tr>
<tr>
<td>3.6</td>
<td>Example of Style assignment</td>
<td>16</td>
</tr>
<tr>
<td>3.7</td>
<td>Example of a Control Template</td>
<td>17</td>
</tr>
<tr>
<td>3.8</td>
<td>Example of data binding</td>
<td>17</td>
</tr>
<tr>
<td>3.9</td>
<td>Example of a Model-View-ViewModel design pattern</td>
<td>18</td>
</tr>
<tr>
<td>4.1</td>
<td>Forces versus distance between vertices</td>
<td>21</td>
</tr>
<tr>
<td>4.2</td>
<td>Snapping of a node to grid point</td>
<td>22</td>
</tr>
<tr>
<td>4.3</td>
<td>Effect of angular resolution criterion</td>
<td>22</td>
</tr>
<tr>
<td>4.4</td>
<td>Effect of edge length criterion</td>
<td>23</td>
</tr>
<tr>
<td>4.5</td>
<td>Effect of line straightness criterion</td>
<td>24</td>
</tr>
<tr>
<td>4.6</td>
<td>Effect of octilinearity criterion</td>
<td>24</td>
</tr>
<tr>
<td>4.7</td>
<td>Effect of balanced edge length criterion</td>
<td>25</td>
</tr>
<tr>
<td>5.1</td>
<td>Conceptual diagram of software architecture</td>
<td>29</td>
</tr>
<tr>
<td>5.2</td>
<td>Class diagram of the Model</td>
<td>29</td>
</tr>
<tr>
<td>5.3</td>
<td>Visual tree of the main map container</td>
<td>30</td>
</tr>
<tr>
<td>5.4</td>
<td>XAML code of DataTemplate for location icon currently in use</td>
<td>35</td>
</tr>
<tr>
<td>5.5</td>
<td>XAML code of a new location icon</td>
<td>36</td>
</tr>
<tr>
<td>5.6</td>
<td>Effect of changes to DataTemplate</td>
<td>36</td>
</tr>
<tr>
<td>5.7</td>
<td>Exposed command property in MapViewModel</td>
<td>37</td>
</tr>
</tbody>
</table>
5.8 Assignment of command to visual element (button) ........................................... 37
6.1 Default user interface for setting up route knowledge ................................. 39
6.2 Visual representation of test data ................................................................. 39
6.3 Visual representation of correct solution of the test exercise ....................... 40
G.1 Evaluation test completion times ............................................................... 57
G.2 Number of clicks to complete the evaluation test .................................... 58
G.3 Mistakes made in evaluation test ............................................................... 58
H.1 Initial solution for Sodor network ............................................................... 59
H.2 Sodor network after force and snap functions applied .............................. 60
H.3 Solution of Sodor network ....................................................................... 60
H.4 Initial solution for Virgin West Coast network ........................................ 61
H.5 Virgin West Coast network after force and snap functions applied ............ 61
H.6 Solution for Virgin West Coast Network after 24 iterations ..................... 62
Chapter 1

Introduction

1.1 Outline

Train crew scheduling is the process of constructing shifts to be assigned to crew members. Input to the problem is a sets of trips and resource, labour agreement and route knowledge constraints. TrainTRACS is a software package developed by Tracsis plc for automating such process. Using the software involves extensive configuration stage one part of which is setting up route knowledge table. Columns of the table represent depots, rows individual route segments and each cell holds boolean value indicating whether drivers of a depot have knowledge of corresponding route segment. It is common for these tables to be very large due to number of route segments involved. Logical correctness of such table is of great importance as it directly influences the solution. Current practice for setting up such data in TrainTRACS is to use graphical user interface containing large table of Y/N values. However this process is very tedious and often involves human error.

1.2 Aim

Aim of this project is investigate data involved in configuration of route knowledge and attempt to replace matrix based user interface currently in use by fully flexible software for interactive visualisation of geographical data using metro style map layout algorithm and colour coding for route segments.
1.3 Objectives

There are three main aspects of the project: software engineering, algorithm and application. Therefore the following objectives are divided into categories corresponding to such aspects.

• Preparation
  – Analyse data
  – Research relevant literature

• Software engineering objectives
  – Produce software architecture model based on Model-View-ViewModel pattern
  – Implement graphical user interface
  – Perform software flexibility evaluation

• Application objectives
  – Gather functional requirements
  – Implement user interaction facilities for setting up route knowledge
  – Design software usability evaluation framework
  – Perform software usability evaluation and analyse the results

• Algorithm objectives
  – Develop algorithm for automatic generation metro style map layout
  – Perform the evaluation of the algorithm

1.4 Requirements

1.4.1 Minimum Requirements

• Interactive user interface capable of:
  – displaying location icons and route segments joining them
  – user interaction to set up route knowledge requirements

• Flexible software architecture model based on Model-View-ViewModel architectural pattern

• Algorithm for automatic generation of metro style map layout.
1.4.2 Possible Extensions

- User interface feature to display junctions
- User interface feature to display some route segments as curves

1.5 Deliverables

The main deliverable of the project is a software package capable of reading in geographical data and producing visual representation of it using automatic metro style map generation algorithm. It should have a graphical user interface to setting up route knowledge matrix using visual representation of data and colour coding of route segments. This should provide users of TrainTRACS crew scheduling software with more intuitive method of inputting data than current practice of large Y and N values matrix.

1.6 Relevance to Degree

This project is particularly relevant to my degree in three aspects:

**Adoption of specific technology (WPF)** During the course of this project I had to learn and adapt Windows Presentation Foundation, technology new to me. This was quite a challenge but the benefit is obvious: I got the practical experience and shifted my way of thinking about user interfaces which is one of the main difficulties in learning WPF.

**Software Engineering** Deliverable of this project is a software package that implements Model-View-ViewModel architectural design pattern. Learning and adapting the concepts of MVVM helped me to look deeper into software engineering concepts such as loose coupling and separation of concerns. I am looking forward to further investigating this pattern.

**Algorithm design** Investigating and adapting metro style map layout algorithms helped me to improve my skills in algorithm design. Particularly in the field of heuristic local search based algorithms.

1.7 Project Schedule

Initially intention was to implement all the code in weeks 5-10. This was to ensure that there is enough time to test the software and produce a documentation for this report. However in practice coding part of the project lasted longer than anticipated. This was due to steep learning curve of technology. Gantt charts for initial and revised project plans are presented in Appendix C.
Chapter 2

Background

2.1 Metro Maps

2.1.1 Overview

A map is a graphic representation or scaled model of a spatial content. They are universal medium for communication independent of language or culture. This is why maps became such a popular tool. The earliest surviving map can be traced back to 1150 BC [7]. Initial objective of a map was to accurately represent geographical data. However it is not always possible and new approaches of visualisation had to be taken.

Metro style maps became popular form of visualisation of geographical data since the beginning of 20th century. This is when inner city railway networks such as London reached the size and complexity too difficult to visualise accurately. Therefore the need to reduce the accuracy in order to increase readability of such maps became apparent. Eventually after the emergence of metro style maps they became an ”interface between the city and its public” [18]. The following sections describe main characteristics, brief history and current practices of automated layout of metro style maps.

2.1.2 Characteristics

There are three main characteristics applicable to metro style maps: shape generalisation, angle generalisation and colour coding [16]. The following paragraphs in this section will describe each of the characteristics.
2.1.2.1 Shape generalisation

The concept of schematic maps using spatial distortion began with straightening on lines. This is particularly useful when it comes to reducing the complexity of visual elements representing map objects. Figure 2.1 illustrates the effect of line straightness feature for lines between three locations.

![Figure 2.1: Segment of map before (above) and after applying shape generalisation](image)

2.1.2.2 Angle generalisation

From the concept of shape generalisation emerged another prominent characteristics of metro style maps - angle generalisation. Since all of the line segments are now straight lines one can restrict their positioning to horizontal, vertical or at 45° angle. This helps to achieve eight-fold symmetry and improve the overall appearance of the map [16]. Figure 2.2 illustrates angle generalisation applied to five nodes.
2.1.2.3 Uniform spacing of locations

Accurate geographical map of a wider area will suffer from some locations being drawn too close to each other while others being spread. This is particularly relevant to maps covering both rural and urban areas. Such maps are either difficult to read or has some location removed in order to visualise data accurately. To overcome this problem uniform spacing of locations is used in metro style maps. Figure 2.3 illustrates the effect of uniform spacing feature.
2.1.3 Brief History

The very earliest railway maps in (such as Liverpool - Manchester) were simply etched onto existing geographical maps that included surface topological features (i.e. lakes and parks). However in mid 19th century construction of nearly 10,000 km of new railways took place in Britain [16]. This led to burgeoning networks growing in size and complexity. Accurate geographic visualisation of such networks resulted in “spaghetti” like maps very difficult to read.

First wave of specialised maps began with exclusion of almost all surface topography. However for more complexed networks (i.e. London) this was not sufficient. Therefore the need for spatial distortion for such maps became apparent. Furthermore without spatial distortion maps covering both urban and rural areas became imposible to draw. Early example of such distortion is London Metropolitan Railway map (figure 2.4) published in 1896. Here locations in top left corner are much further apart than the ones in bottom right corner.

The next step in the evolution of metro style maps came with the introduction of straight lines. Earliest maps to use this feature were published in mid 1920’s. In 1929 this feature was adapted by Geogre Dow in his London and North Eastern Railway map. Many historians believe that Dows work inspired Harry Beck to produce probably the most famous schematic of such style for London Underground in 1933 (figure 2.5). Beck extended the concept of straight lines (shape generalisation) position lines vertically, horizontally or at 45° angle. After this pioneering work, many transit authorities worldwide imitated the diagrammatic look for their own networks. Early maps of the Berlin U-Bahn, Boston T, Paris Mtro and New York Subway also exhibited some elements of the diagrammatic form.

After Beck’s 1933 London Underground schematic, many others appeared following similar ideas. However, there was very little innovation and new ideas tended to be minor cosmetic design tweaks. Thanks to the ideas introduced by both Dow and Beck we still use schematics with the same features more han 70 years later [16].
Figure 2.4: Map of London Metropolitan Railway Network (1986)

Figure 2.5: Metro Map for London Underground by Harry Beck (1933)
2.1.4 Automated Layout

Manual drawing of metro style maps having characteristics described in previous section is still common. However it can very challenging for more complexed networks. Nollenburg [13] proves the NP-completeness of metro map layout problem involving angle generalisation. Hence significant effort was made by researchers to develop algorithms based on stochastic and exact techniques to automate the layout process. Some of them are reviewed in this section.

2.1.4.1 Multicriteria Hill Climbing

Stott [16] suggests multicriteria hill climbing technique based on movement of individual nodes to more optimal locations according to six aesthetic criteria:

- **Angular resolution**: enforcing angle to be as large as possible between adjacent edges
- **Edge length**: enforcing nodes to be spaced as evenly as possible
- **Balanced edge length**: enforcing edges adjacent at nodes of degree 2 to be of approximately the same length
- **Edge crossings**: minimising the number of unnecessary edge crossings
- **Line straightness**: enforcing edges forming a line to be collinear either side of each node
- **Octilinearity**: enforcing lines to be drawn horizontally, vertically or at 45 degree

It also makes use of the identification of clusters (by finding overlength edges) and moving them using similar rules as for individual nodes. Extensive empirical evaluation was performed to show that such hill climbing algorithm produces maps comparable to hand drawn ones. This methodology will be further investigated in Chapter 4.

2.1.4.2 Force-Based Algorithms

Hong at al [8] investigated various combinations of directed spring based algorithms:

- **Graph Embedder (GEM) algorithm** proposed by Frick et al [5] based on the observation that cooling systems give better results than methods relying on gradient descent. Temperature in this case is the maximum distance a distance can be moved when being updated.

- **Force directed algorithm preserving edge crossing properties** by Bertault [1]. It uses the analogy with physical model by representing nodes by magnetic rings repelling each other and edges by springs connecting rings. In this force model, any two rings repel each other, and two rings linked by a spring are attracted until the natural length of the spring is achieved. The equilibrium state of the physical system gives the position of the nodes in the drawing.
• Magnetic spring algorithm by Sugiyama and Misue [17] which uses similar ideas to the algorithm by Bertault [1].

They showed that spring based algorithms can be used to draw metro style maps comparable to hand drawn ones in reasonable amount of time.

2.1.4.3 Other Heuristics

Dwyer at al [3] proposes heuristic algorithm the main strategy of which is to:

• Partition the path into blocks of points
• Fit the points with a set of line segments aligned with a limited set of possible directions using least-squares regression method
• Merge partitions back together

They developed algorithm capable of producing schematic map for Sidney Cityrail of reasonable quality showing that altering the definition of quality of fit can be used to create simpler and faster algorithms.

2.1.4.4 Mixed Integer Programming

Mathematical programming approach suggested by Nollenburg and Wolff [14] formalises metro style requirements to a set of linear constraints. They performed extensive comparison between their proposed MIP and heuristic techniques suggested by Hong [8] and Stott [16] showing that their approach deals better with octilinearity constraints and avoids the problem of getting stuck in local minima. However main disadvantage of MIP approach is running time which is significantly longer.

2.1.4.5 Summary

A number of techniques for automated layout of metro style maps were investigates. Most of them are quite difficult to implement and experiment with in a reasonably short time. Multicriteria hill climbing algorithm proposed by Stott [16] was found to be attractive because of clearly defined objectives and methodology. Hence this technique was chosen to be further investigated and adapted to the requirements of this project.
Chapter 3

Technology

3.1 Windows Presentation Foundation

3.1.1 Overview

Microsoft Windows operating systems provide a set of application programming interfaces (API) which has been the most direct mechanism with which an application can interact with Windows. Major versions of Windows API are Win16, Win32 and Win32 for 64-bit Windows and include the following functional groups [15]:

- **Graphics Device Interface (GDI)** Provides the facilities for an application to generate graphical output to displays, printers, and other devices.

- **User Interface (UI)** Provides the mechanism for managing windows and controls (such as buttons, text boxes, etc) in an application and input from devices such as the mouse and keyboard.

Since the release of Windows 1.0, the GDI and the UI services of the Windows API have provided a decent graphics platform for Windows applications. Many applications we use on a day-to-day basis are built on this foundation. With the increase in number of computers and the increasing availability of the Internet access, the demand for rich visual experiences has dramatically increased. Facing this new demand, Microsoft has invested heavily in providing such an experience as part of the Windows application programming interface - .NET Framework 3.0 which was an important milestone for developing Windows applications [15]. Built on top of the .NET 2.0, .NET 3.0 is a set of managed APIs that provide enhanced functionality for messaging, work flow, and presentation. The following sections will be focused on major features of WPF (Windows Presentation Foundation), graphical subsystem of .NET 3.0 framework.
3.1.2 XAML

Typical development of a graphical user interface for a typical Windows-based application encompasses the following general tasks:

- Define and set up controls on a form
- Hook data-bound controls to data sources, typically a database or related data access objects
- Configure control events so that user interaction triggers application logic to manipulate data and/or the UI

In most of the standard GUI development practices the above goals are achieved in imperative manner. That is, the definition of controls, their binding, and user interaction is included within code as statements which modify the state of the application, and are typically compiled directly into an executable format [15]. XAML (Extensible Application Markup Language) is an XML based markup language used to instantiate .NET objects. XAML provides a declarative mechanism (as opposed to imperative one) to construct WPF user interfaces. For example consider the following XAML code snippet in figure 3.1. Here Label and a Button are declared as part of the Grid which is embedded in a Window.

```xml
<Window x:Class="MyXamlTest.Window1"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="XamlTestWindow" Height="300" Width="300">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="50px" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Label Grid.Column="0" Grid.Row="0">Click This Button:</Label>
    <Button Grid.Column="0" Grid.Row="1" DockPanel.Dock="Top"
            Name="clickButton">Click Me!</Button>
  </Grid>
</Window>
```

Figure 3.1: Declarative definition of a user interface in XAML

Although XAML provides a clear declarative model for developing WPF applications, it possible to achieve the same purely in code. For example figure 3.2 shows the above piece of XAML code can be reproduced in C#.
In most situations creating the exact same application using C# code requires a significantly greater amount of statements than its XAML counterpart [15]. WPF will actually create a source file at compile time for each XAML file. The code generated will be very similar to the example in figure 3.2. However, the source remains as a XAML file which can be developed separately from
application logic.

3.1.3 Visual and Logical Trees

A fundamental concept of WPF is the notion of element trees, which represent the visual elements comprising a graphical user interface. Two element trees exist within a WPF application: the logical tree and the visual tree. Difference between the two is best explained through the example. Consider the following XAML code snippet in figure 3.3.

```xml
<Window x:Class="MyXamlTest.ElementTrees"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="ElementTrees" Height="300" Width="300">
    <StackPanel>
        <Label Name="nameLabel">Please Enter Your Name:</Label>
        <TextBox Name="nameTextBox" Margin="2px"></TextBox>
        <Button Name="submitButton" Margin="2px">Submit</Button>
    </StackPanel>
</Window>
```

Figure 3.3: XAML code for the example of element trees

Logical Tree The logical tree (Fig 3.4) outlines the one to one mapping of the nested XAML elements declared in the code snippet to their appropriate classes [12]. WPF makes use of the logical tree when looking up for UI element names, routing events and inheriting properties.

![Logical Tree Diagram](image)

Figure 3.4: Example of a logical tree

Visual Tree A visual tree (figure 3.5) is an expanded version of the logical tree. It breaks elements down into smaller pieces. It allows us to do two useful things: alter one of the elements in the visual tree using styles and apply templates to controls. Both of these mechanisms will be described in the following sections. In order to demonstrate the difference between logical and visual trees the diagram in figure 3.5 elements belonging to logical tree are coloured using different shade.
3.1.4 Styles

WPF styling mechanism allows to uniformly apply a group of properties to controls in order to define a desired appearance. The mechanism is similar to CSS (Cascade Styling Sheets) and its usage for HTML (Hyper Text Markup Language) elements. Styles provide a centralized location to define the visual appearance for the controls. Moreover styles propagate those attributes to any element that inherits the style. Example in figure 3.6 illustrates how style with two setters (for background and foreground colours) is defined and assigned to a TextBox.
3.1.5 Templates

Styling provides a decent mechanism for changing the appearance of a control. However they are limited in the way that they depend on properties provided by the control itself. Templates allows to completely change the appearance of some particular visual element (by overriding its visual tree). In practice it is common to have most of the user interface declared as templates. There are three types of templates in WPF: DataTemplates, ControlTemplate and ItemsPanelTemplate. The following sections will describe each type of templates.

**DataTemplates** DataTemplates are used to create visualisation of non-visual objects, such as business objects. Such objects are often displayed within a control like ListBox, ComboBox or ListView. Default data template is a simple TextBlock bound to string representation of a data object (retrieved by ToString() method). Hence data templates allows to define a visual tree of a visual item representing data object. For example data object could be displayed as a StackPanel containing photo followed by some text.

**ControlTemplates** ControlTemplate provides a mechanism to define visual representation of a UI control, such as a Button or TextBox. For example in order to make a button look like an ellipse we could simply define such ControlTemplate as illustrated in figure 3.7.
Figure 3.7: Example of a Control Template

**ItemsPanelTemplate**  For visual elements of type FrameworkContentElement (the ones containing visual elements for data objects such as ListBox, ComboBox, ListView) ItemsPanelTemplate allows to create the define the appearance of the container of visual elements. This is particularly important for cases like ComboBox. Here ControlTemplate can be used to define the look of the main control box while ItemsPanelTemplate can be used to define the appearance of the box that contains the list of items and pops down when ComboBox is clicked.

3.1.6 Data Binding

Typical user interface exposes properties of one or more data objects. Therefore when such property changes the user interface has to be updated and vice versa. WPF data binding provides the way to avoid explicit update of user interface and data objects. Databinding can be unidirectional (source to target or target to source), or bidirectional. DataContext property of a visual element is assigned a reference to some data object in order to bind their properties. Databinding of individual properties is typically done in XAML by using the `{Binding}` markup extension. Figure 3.8 shows how Name and Surname properties of a data object are bound to Text properties of two TextBoxes.

```
...  
<Label Content="Name:" />
<TextBox Text={Binding Name} />

<Label Content="Surname" />
<TextBox Text={Binding Surname} />
...
```

Figure 3.8: Example of data binding

3.1.7 Commands

Commands in WPF are used similarly as event handlers and have several purposes. The first one is to separate the declaration of the object that invokes a command from the logic that executes the command. This allows for multiple sources to invoke the same command logic, and it allows the
command logic to be customized for different targets. Another purpose of a command is to indicate whether it is allowed to be executed. This is particularly useful when it comes to enabling/disabling user interface elements that calls the command.

3.2 Model-View-ViewModel Architectural Design Pattern

Developing graphical user interface that exposes the required data and at the same time meets stylistic requirements can be very difficult task [4]. Furthermore desire of separation of concerns [9] can make it ever harder because user interface framework does not allow implementation of a particular architectural design pattern. Windows Presentation Foundation framework and Model-View-Presenter pattern [2] gave birth to Model-View-ViewModel (MVVM) pattern which was proposed by John Grossman in 2005 [10]. It main goal is separation of functional requirements by making use of specific WPF features like data templates, data binding and commands.

The View (graphical user interface and its behaviour) is bound to properties exposed by a ViewModel (abstraction of a view), which, in turn, exposes data contained in data model objects and other state specific to the view. The bindings between View and ViewModel are easy to construct in WPF through XAML markup extensions.

If property values in the ViewModel change, those new values automatically propagate to the view via data binding. Moreover this allows to implement and perform unit tests for ViewModels to test the validity both data model and user interface [11]. The following paragraphs will describe the main components of Model-View-ViewModel architectural design pattern through the example in figure 3.9. Here a Person object is bound to PersonView through PersonViewModel.

![Diagram of Model-View-ViewModel design pattern](image)

**Figure 3.9: Example of a Model-View-ViewModel design pattern**

**Model** The concept of model for MVVM emerges from Model View Controller pattern. It is the data or business logic, completely independent from user interface and stores the state and does the processing of the problem domain. Example in figure 3.9 illustrates how properties of PersonViewModel are bound to properties of Person object.
**View**  The View in MVVM consists of the visual elements, the buttons, windows, graphics and more complex controls of a GUI. The View is usually defined declaratively. Example in figure 3.9 shows two way binding between properties of PersonViewModel and values of user interface elements. Here Save button is bound to SaveCommand exposed by ViewModel.

**ViewModel**  The term means ”Model of a View”, and can be thought of as abstraction of the view. It also provides a specialisation of the Model that the View can use for data-binding. In this latter role the ViewModel contains data-converters that convert Model types into View types, and it contains Commands the View can use to interact with the Model. Example in figure 3.9 a class Person exposes a height property in centimeters. Corresponding PersonViewModel class is then exposing another property of height, but in this case in inches (acting as a value converter). PersonViewModel is also serves the purpose of a container for SaveCommand (bound to Save button). Since in this example binding between properties Model and ViewModel is one way only (from ViewModel to Model) SaveCommand updates the model when Save button is clicked.

### 3.3 Summary

Requirements from the company for this project are that the software should be flexible and follow high standards of industry as well as have a decent look-and-feel. The choice of WPF as a graphical framework stems from such requirements. This leads to the choice of architectural design pattern of Model-View-ViewModel. Implementation of this pattern is described in Chapter 5.
Chapter 4

Algorithm

4.1 Overview

This section will introduce to the implementation of multicriteria hill climbing algorithm for automated metro style map layout problem proposed by Stott [16]. It is based on 5 aesthetic criteria emerging from the characteristics of metro style maps described in Chapter 2 as well as two node movement rules. Pseudocodes for mode movement and iteration procedures are made available at the end of chapter.

4.2 Preprocessing

Initial embedding Initial allocation of node positions is based on their actual geographical coordinates. However this embedding can potentially cause problems when snapping nodes to grid points because some locations might end up too close to each other. Therefore initial step is to apply force-based algorithm to prepare the map for snapping.

Directed force based algorithm The force based simulated annealing algorithm used here was initially developed by Reignolds and Fruchterman [6]. The main idea is to mimic the physical model based on electrostatic attraction. Attractive and repulsive forces are calculated for each of the locations according to the distances between vertices (figure 4.1): $f_r = \frac{k^2}{d^2}$, $f_a = \frac{d}{k^2}$ (where d is the distance between vertices and k is the optimal distance). This forces connected locations to be closer to each other but not too close (the optimal distance k is calculated using formula: $k = \sqrt{\frac{area}{N}}$, where N is the total number of vertices).
Snapping to grid points  The next step of preparation is to snap locations to grid points as pictured in figure 4.2. This step is very important and affects the outcome of the following iterations. Before snapping nodes to grid it is important to determine the size of the grid cell and radius of search space for node placement. Clearly network with more locations requires a denser grid. The steps involved in snapping process are as follows:

- First step is to order the list of locations by their distance to the nearest grid point. This helps to avoid edge crossings when snapping nodes since snapping node to already occupied grid intersection point is not permitted.

- Then for each of the nodes we consider a number of grid points calculating penalty values for them using the function described below picking the grid point with lowest penalty value.

4.3 Aesthetic Criteria

Hill climbing optimiser should minimise a particular objective function. This function is composed of five aesthetic criteria based on the ones proposed by Stott [16]. They emerge from the characteristics of the metro style maps described in Background chapter. The goal for the algorithm is to minimise each of the criteria. However they differ in range of magnitude. Therefore care has to be taken when
determining the weightings for each of the criterion. The following sections describe each of the aesthetic criterion.

### 4.3.1 Angular Resolution

Particularly small angles are not desired in metro style maps. Especially for locations with multiple passing lines. Angular resolution criterion is incorporated in order to increase the sum of angles between edges for each vertex. Figure 4.3 illustrates the effect of angular resolution criterion. Here outer locations get situated around the central one to form lines of maximised angles. Formula for calculating penalty of angular criterion for vertex $v$ of degree $d(v)$ and set of adjacent edges $E(v)$ is:

$$C_1 = \sum_{(e_1, e_2) \in E(v)} \left\| \frac{2\pi}{d(v)} - \theta(e_1, e_2) \right\|$$
4.3.2 Edge Length Criterion

This criterion corresponds to the feature of metro style maps regarding even spacing of locations. This is the only aesthetic criterion having a parameter which is integer number $l$ representing optimal spacing between two points in grid spacing units. Penalty value is assigned for edges that are longer or shorter than $lg$ where $g$ is the grid spacing. The effect of such criterion is presented in figure 4.4. Here optimal distance is twice the grid spacing. Location in top left corner is brought closer to the remaining 3 to minimise the penalty value. Formula for calculating penalty value for edge length criterion of vertex $v$ with a set of adjacent edges $E(v)$ and optimal edge length $d$ is:

$$C_2(v) = \sum_{e \in E(v)} \left\| \frac{\|e\|}{d} - 1 \right\|$$

![Figure 4.4: Effect of edge length criterion](image)

4.3.3 Line Straightness

It is very common to have location with two neighbours either side. It usually takes place where railway line passes through a particular location. It is desired to have such segments of route in as straight lines as possible. Hence vertices of degree two are penalised for angles between lines. The effect of such criterion is displayed in figure 4.5. Here location in the middle is aligned with the remaining two to minimise the angle between lines. Calculation of the value for line straightness criterion for vertex $v$ with two adjacent edges $e_1$ and $e_2$ is simply the angle between them:

$$C_3(v) = \theta(e_1, e_2)$$
4.3.4 Octilinearity

Requirement of lines to be placed horizontally, vertically or at 45° angle corresponds to the octilinearity criterion. Figure 4.6 illustrates the effect of this criterion. Here bottom and top vertices are brought to positions to form angles of 90° and 45° respectively. Formula for calculating octilinearity criterion penalty value for vertex $v$ with neighborhood $N(v)$ is:

$$C_4(v) = \sum_{v' \in N(v)} 4\theta(v, v') \mod \pi$$

Figure 4.5: Effect of line straightness criterion

Figure 4.6: Effect of octilinearity criterion
4.3.5 Balanced Edge Length

Similarly as line straightness this criterion is only applied to vertices of degree two. Value for such criterion is calculated by taking ratio of longer line over smaller one. Figure 4.7 illustrates the effect of balanced edge length criterion. Here middle location is brought in the middle of the other two. Formula for calculating balance edge length criterion for vertex $v$ with two adjacent edges $e_1$ and $e_2$ is simply the difference of their lengths:

$$C_5(v) = ||e_1|| - ||e_2||$$

Figure 4.7: Effect of balanced edge length criterion

4.4 Node Movement

Movement of particular node depends on several rules breaking of which is not allowed as well as penalty value calculated using five aesthetic criteria described in previous section. This section will describe the main node movement rules and the actual node movement procedure.

Geographical Relations Rule Despite the fact that metro style maps are an approximation of the real geographical data retention of geographical relations is very much desired. For example we would like to have Manchester west of Leeds and not the other way round. For that reason such geographical relation is checked between the node that is currently being moved and all its neighbouring ones.

Occlusion Rules Occlusions as well as unnecessary crossings of lines is not desirable in any map. Therefore the following four situations are prevented when moving a particular node $A$:

- Edge incident to $A$ crossing or lying on top of any other edge
- Edge incident to $A$ crossing any other node
• A lying on top of any other edge

• A lying on top of any other node

4.4.1 Node Movement Procedure

Movement of node occurs after consideration of potential positions within given radius (multiple of grid spacing $g$). Each potential position is checked for violation of node movement rules and in case of being legal gets assigned a penalty value. Such penalty value is then compared against the best penalty value found so far. If the new penalty value for a particular position is smaller than the best one it gets recorded. After all possible positions are considered the lowest penalty one is assigned to that particular node. Below is the pseudocode describing node movement procedure.

\begin{verbatim}
proc MoveNode(v)
    bestPenalty ← CalculateNodePenalty(v)
    currentX ← v.x
    currentY ← v.y
    bestX ← currentX
    bestY ← currentY
    for i := −radius to radius do
        for j := −radius to radius do
            x ← currentX + i × g
            y ← currentY + j × g
            if CheckGeographicalRelationsRule(v, x, y) = true
                if CheckOcclusionRules(v, x, y) = true
                    penalty ← CalculateNodePenalty(v)
                    if penalty ≤ bestPenalty
                        bestPenalty ← penalty
                        bestX ← x
                        bestY ← y
                fi
            fi
        od
    od
    v.x ← bestX
    v.y ← bestY
\end{verbatim}
4.5 Iterations

Execution of the algorithm begins by applying force based algorithm to the set of vertices to spread them out. Then snapping procedure is called for the set of vertices to make initial embedding of the map. Now the hill climbing optimiser is called with a given maximum number of iteration and tolerable error (difference between penalty values after two subsequent iterations) value. Experimental results showed that changing the order in which nodes are moved helps to avoid local minima therefore at the beginning of each iteration list of nodes is being shuffled. After node positions are updated the radius gets reduced by cooling function (in this case radius is inversely proportional to the fraction of iterations currently performed).

\[\text{begin}\]
\begin{align*}
&\text{ApplyForce}(V,E) \\
&\text{SnapNodes}(V) \\
&\text{bestPenalty} \leftarrow \text{CalculateTotalPenalty}(V,E) \\
&\text{currentPenalty} \leftarrow \text{bestPenalty} \\
&\text{running} \leftarrow \text{true} \\
&\text{iteration} \leftarrow 0 \\
&\text{while} \ \text{running} = \text{true} \ \text{do} \\
&\quad \text{Shuffle}(V) \\
&\quad \text{for } v \in V \ \text{do} \\
&\quad\quad \text{MoveNode}(v) \\
&\quad \text{od} \\
&\quad \text{penalty} \leftarrow \text{CalculateTotalPenalty}(V,E) \\
&\quad \text{difference} \leftarrow \text{penalty} - \text{currentPenalty} \\
&\quad \text{iteration} \leftarrow \text{iteration} + 1 \\
&\quad \text{radius} \leftarrow \text{Cool}(\text{radius}, \text{iteration}) \\
&\quad \text{if} \ \text{iteration} = \text{maxIteration} \ \vee \ \text{difference} \leq \text{tolerance} \\
&\quad\quad \text{running} \leftarrow \text{false} \\
&\quad \text{fi} \\
&\text{od} \\
&\text{end}\]
\[\text{end}\]
Chapter 5

Implementation

5.1 Overview

Three major objectives of this project are to:

- produce an interactive graphical user interface for visualising geographical data
- algorithm for automated layout of the metro style maps
- software architecture that would provide a decent mechanism for the data model to interact with the user interface and the algorithm (specifically addressing route knowledge table set up)

Description of the algorithm was presented in previous chapter. In this chapter algorithm is part of the Model and will not be explored. Software was designed and implemented following guidelines proposed by Model-View-ViewModel architectural design pattern. High level view of the components of MVVM is presented in figure 5.1. The following sections will be dedicated to each constituent part of such architecture. Screenshot of the user interface is presented in Appendix F.
5.2 Model

Model is the simplest part of the software. Investigation of functional requirements and available data led to the model presented in figure 5.2. Here there are two main classes representing map objects: Location and Leg (or segment of route). Both of them are exposed to the algorithm through IMapObject interface. Location is assigned a Coordinates object containing easing and northing coordinates of the geographical location it represents. Location may have a Depot object assigned which is a collection of Links (sub-depots). Links are used as annotation for Legs. Each Leg can be assigned to multiple links and multiple Legs can be assigned to a particular Link. UML class diagram of the data model is presented in figure 5.2.
5.3 View

In the context of this software View encompasses graphical user interface and its behaviour. This section will describe individual components of the view.

5.3.1 Map Container

Implementation of main container holding map objects makes use of the functional features of a ListBox; it is capable of containing multiple items and providing a selection functionality for them. Windows Presentation Foundation styling and templating mechanisms allow to completely rewrite the appearance and behaviour of a user control. Hence map container is implemented as a ListBox with custom ControlTemplate, ItemsPanelTemplate and DataTemplates for data objects. Figure 5.3 illustrates the visual tree of main map container.

![Figure 5.3: Visual tree of the main map container](image)

To provide user with zooming capabilities of map container visual tree begins with a ScrollViewer (wrapper control providing scrolling capabilities to the content). Content of the ScrollViewer is a simple grid containing BusyAnimationLayer canvas stacked on top of MainListBoxCanvas container. This is to display a busy animation and prevent the user from interacting with the map when algorithm is running. MainListBoxCanvas contains ItemsPresenter defined by custom ItemsPanelTemplate. This canvas holds visual elements for data objects defined by DataTemplates.
Four modes of mouse pointer are provided to interact with the map container:

- **Navigate.** Implementation drag-to-scroll functionality.
- **Move.** Implementation of drag-and-drop functionality for visual items representing Locations.
- **Zoom.** Provides functionality to draw rectangle to indicate the desired zoom region.
- **Select.** Provides functionality to draw rectangle to select lines representing Legs.

### 5.3.2 Map Objects

Two types of map objects are displayed Locations and Legs. They are exposed to view through their ViewModels. Both of them have DataTemplates (contained in DataTemplateResourceDictionary) in order to be displayed on map.

Location is represented by a horizontal StackPanel containing Ellipse followed by TextBlock wrapped in a Border. Ellipse is assigned IsNode attached property exposed by ListBoxItemIdentificationHelper class. This is to keep track of visual elements representing nodes by storing references to them in dictionary using unique identifier exposed by data context (which in this case is a ViewModel of a Location).

Leg is represented by LegLine, custom implementation of FrameworkElement. It has OnRender method overridden to draw a line or multiple parallel lines between two end points. LegLine object representing Leg is assigned IsEdge attached property of ListBoxItemIdentificationHelper. LegLines data context (ViewModel of Leg) has Ids of start and end points exposed which allows ListBoxItemIdentificationHelper to identify visual elements representing Locations and bind their screen coordinates to end points of LegLine.

### 5.4 ViewModel

ViewModel part of the software includes ViewModel classes encapsulating data model items and commands exposed by them. Following are the description of main ViewModels for map, location and leg (segment of route).

#### 5.4.0.1 ViewModel of Map

MapViewModel is used as the data context of the user control representing map. It exposes a MapObjects collection as a source of items for map container as well as collection of Depots to keep track which Locations were assigned a depot attribute and present such information to the View. MapViewModel also exposes a number of View related properties and commands to for the View to interact with the Model.
<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background</td>
<td>Brush</td>
<td>Brush used to colour the main container of the user control representing map.</td>
</tr>
<tr>
<td>BoxBackground</td>
<td>Brush</td>
<td>Brush used to colour the box containing label of the Location.</td>
</tr>
<tr>
<td>BoxStrokeThickness</td>
<td>Double</td>
<td>Fractional number representing thickness of the line around the label Location.</td>
</tr>
<tr>
<td>ContainerSize</td>
<td>Double</td>
<td>Fractional number representing size of the of the map canvas.</td>
</tr>
<tr>
<td>FontFamily</td>
<td>FontFamily</td>
<td>C# FontFamily object bound to the Locations labels font family property.</td>
</tr>
<tr>
<td>FontSize</td>
<td>Double</td>
<td>Font size of the Location label.</td>
</tr>
<tr>
<td>GridSize</td>
<td>Double</td>
<td>Fractional number representing the number of grid points in any direction.</td>
</tr>
<tr>
<td>LegThickness</td>
<td>Double</td>
<td>Fractional number representing thickness of a line representing Leg.</td>
</tr>
<tr>
<td>AnimationVisibility</td>
<td>Bool</td>
<td>Indicates whether to show loading animation.</td>
</tr>
<tr>
<td>ShowBoxes</td>
<td>Bool</td>
<td>Indicates whether to show boxes around Location labels.</td>
</tr>
<tr>
<td>ShowGridLines</td>
<td>Bool</td>
<td>Indicated whether to draw grid lines on the user control representing map.</td>
</tr>
</tbody>
</table>

Table 5.1: Properties exposed by MapViewModel

5.4.0.2 ViewModel of Location

Main responsibility of LocationViewModel is to wrap the Location object and act as a medium of communication between Location and visual element representing it. It contains DepotViewModel property which in the case of Location being converted to Depot gets assigned a reference to ViewModel of Depot object. It communicates with ListBoxIdentificationHelper class of the View through INode interface. Properties exposed by LocationViewModel are presented in table 5.2
Table 5.2: Properties exposed by LocationViewModel

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Location</td>
<td>Location object wrapped by LocationViewModel</td>
</tr>
<tr>
<td>X</td>
<td>Double</td>
<td>Relative X coordinate (ranging from 0 to 1)</td>
</tr>
<tr>
<td>Y</td>
<td>Double</td>
<td>Relative Y coordinate (ranging from 0 to 1)</td>
</tr>
<tr>
<td>Colour</td>
<td>Brush</td>
<td>Solid colour brush to be used to colour visual item representing location</td>
</tr>
<tr>
<td>Diameter</td>
<td>Double</td>
<td>Diameter of the round icon represented by ellipse</td>
</tr>
<tr>
<td>LabelTransform</td>
<td>RenderTransform</td>
<td>Transformation applied to label box (to avoid occlusions)</td>
</tr>
<tr>
<td>ParentContainerSize</td>
<td>Double</td>
<td>Size of the parent container</td>
</tr>
<tr>
<td>ViewX</td>
<td>Double</td>
<td>Actual screen coordinate X of visual element (X * ParentContainerSize)</td>
</tr>
<tr>
<td>ViewY</td>
<td>Double</td>
<td>Actual screen coordinate Y of visual element (Y * ParentContainerSize)</td>
</tr>
<tr>
<td>NodeId</td>
<td>String</td>
<td>Id of node exposed through INode interface</td>
</tr>
<tr>
<td>DepotViewModel</td>
<td>DepotViewModel</td>
<td>DepotViewModel wrapping Depot object assigned to Location</td>
</tr>
</tbody>
</table>

5.4.0.3 ViewModel of Leg

LegViewModel wraps Leg object and exposes its properties to the View. It communicates with ListBoxIdentificationHelper class of the View through IEdge interface which forces them to expose unique identifiers of two end point locations. Table 5.3 lists properties exposed by LegViewModel.

Table 5.3: Properties exposed by LegViewModel

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leg</td>
<td>Leg</td>
<td>Leg object wrapped in LegViewModel</td>
</tr>
<tr>
<td>Colour</td>
<td>Brush</td>
<td>Brush to be used to colour the line representing Leg</td>
</tr>
<tr>
<td>Thickness</td>
<td>Double</td>
<td>Thickness of the line representing Leg</td>
</tr>
<tr>
<td>IsSelected</td>
<td>Double</td>
<td>Indicates whether a particular Leg is currently selected</td>
</tr>
<tr>
<td>StartNodeId</td>
<td>String</td>
<td>Unique identifier of the start location (exposed through IEdge interface)</td>
</tr>
<tr>
<td>EndNodeId</td>
<td>String</td>
<td>Unique identifier of the end location (exposed through IEdge interface)</td>
</tr>
</tbody>
</table>

Table 5.3: Properties exposed by LegViewModel

33
5.5 Review of Software Flexibility

One of the aims of this project was to produce software capable of performing its functions as well as being fully flexible and extensible. To demonstrate such qualities of software and justify the design policy two scenerios were user: replacement of data template and introduction of a new command.

5.5.1 Replacement of Data Templates for Visual Elements

Appearance of visual items representing icons is defined in DataTemplate resource dictionary purely in XAML. This allows to alter visual appearance of software without any need to touch or compile the code. It means that graphics designer can now work with XAML independently from programmer(s) working on application logic side. Moreover there exist a number of software tool to aid graphics designers to work with XAML in WYSIWYG manner. Therefore in terms of modifying the appearance of this software is fairly easy thing to do. This can be illustrated by modifying DataTemplateResourceDictionary.xaml file to change the way program displays location icons. Currently location icon is represented by StackPanel containing Ellipse followed by TextBlock with a visual transformation applied. XAML code snippet defining such appearance is presented in Fig. 5.4.
We can change the look of the location to be a rectangular box with a title text inside it by simply changing XAML code as presented in Fig. 5.6. Important bit is setting ListBoxItemIdentificationHelper.IsNode dependency property as true so that ListBoxItemIdentificationHelper class can treat it as visual item representing node. The effect of such change is presented in Fig. 5.6. Since XAML code can be developed independently from the rest of software this choice of GUI framework helps to make it more flexible.
5.5.2 Introduction of a New Command

MapViewModel class is responsible for exposing the commands of the program to be bound to UI elements of the MapView. UndoRedoContext class and IUndoableCommand interface allows us to implement undoable commands. The following exercise will illustrate how custom behaviour of the program can be introduced by implementing undoable command and exposing it to the view.

We will implement the command which cleans the map from locations that are not connected by any legs (nodes of degree one). In order to do that we write a class based on RelayCommand provided by Model-View-ViewModel framework and implementing IUndoableCommand interface as presented in Appendix E.

- UndoRedoContext is a singleton class responsible for
  - exposing Undo and Redo commands to the view
  - holding the undo and redo command stacks

- IUndoableCommand interface is responsible for
  - exposing Undo and Redo methods to be used by UndoRedoContext

In order for this command to be accessible to the view we need to expose it as a property in MapViewModel (figure 5.7). Finally we can bind this command to visual elements within the view as
depicted in fig. 5.8.
This example illustrates the new commands can be introduced and centralised within the ViewModel.

```csharp
public RelayCommand RemoveDisconectedLocationsCommand
{
    get { return new RemoveDisconectedLocationsCommand(this); }
}
```

Figure 5.7: Exposed command property in MapViewModel

```csharp
<Button Content="Remove Disconnected Locations"
        Command="{Binding RemoveDisconectedLocationsCommand}" />
```

Figure 5.8: Assignment of command to visual element (button)
Chapter 6

Evaluation

6.1 Evaluation of the software usability

6.1.1 Evaluation Framework

To evaluate and compare the usability of software the evaluation framework consisting of main view of the visualisation software and user interface mimicking the behaviour of matrix based interface (figure 6.1) for setting up route knowledge was developed. Human subjects were asked to set up route knowledge matrix using both methods. System recorded amount of time and number of mouse clicks it took to complete the task as well as the number of mistakes left.

For this exercise locations and routes from railway network of fictional island of Sodor (from Thomas The Tank Engine). The data set includes 41 locations, 41 route segments (figure 6.2). Document explaining the process of this test is presented in Appendix D. The task is to convert 9 of the given locations to depots and for each of 9 depots assign the correct set of route segments to set up its route knowledge. Visualisation of the correct solution is presented in figure 6.3

The aim of such experiment is to check the following hypothesis:

- **Hypothesis 1.** It takes less time to complete the above task using map based interface than usual matrix based one

- **Hypothesis 2.** It take fewer mouse clicks to complete the above task using map based interface than usual matrix based one

- **Hypothesis 3.** It is less likely to make mistakes when setting up route knowledge using map based interface than usual matrix based one
Figure 6.1: Default user interface for setting up route knowledge

Figure 6.2: Visual representation of test data
Figure 6.3: Visual representation of correct solution of the test exercise

6.1.2 Evaluation Results

Test was presented to 18 human subjects. All of them completed the tasks. The following are the implications that have affected the outcome of evaluation:

- Exercise was presented in the textual form more similar to the matrix based dialog. Since the route knowledge to set was presented as a bullet points for each depot it required more effort to identify the particular route segments on map compared to matrix.

- Human subjects did not have the knowledge of the geography of the network. Train planners usually have a decent knowledge of geography of the network. Moreover they often use a geographical map in this process which could make using map based interface a lot easier.

- Segments of route were presented in alphabetical order. In real world this is not always the case. However according to feedback from people who did the test this significantly affected the outcome of the test.

- In this exercise the matrix was relatively small (9 by 41). In real world this is not very realistic and the matrix is a lot bigger. This makes it more difficult to use.
• Since using matrix based dialog setting up route knowledge using matrix based interface. Therefore in this exercise each keyboard button stroke was counted as a mouse click.

On average each subject:

• Took 536 seconds to complete the task using map based interface
• Took 543 seconds to complete the task using matrix based interface
• Took 148 mouse clicks to complete the task using map based interface
• Took 166 mouse clicks to complete the task using matrix based interface
• Made 0.17 mistakes using map based interface
• Made 0.72 mistakes using matrix based interface

On average subjects of test took slightly less time to complete the task using map based interface then matrix based one. However these results were affected by implications listed above which limits their accuracy. Nevertheless results and implications confirm the first hypothesis that using metro style map based interface to set up route knowledge takes less time.

Number of clicks metrics is quite similar for both methods. For matrix based interface the number of keyboard strokes was taken into consideration while metro style based interface does not involve keyboard usage at all. On average map based interface required less keyboard/mouse strokes. This confirms the second hypothesis.

In total subjects made 3 mistakes using map based interface and 13 using matrix based one. This clearly confirms the third hypothesis that visual inspection of the route knowledge allows users to spot mistakes more easily.

Results are presented in Appendix G.

6.2 Evaluation of the Algorithm Performance

Two data sets were used for the evaluation of the algorithm: fictional network of Sodor island (from Thomas The Tank Engine TV series) and real world example of Virgin West Coast network. The aim of such tests is to produce solutions for the two exercises.

6.2.1 Fictional Sodor network

This network includes 41 locations and 41 route segments connecting them.

Trial and error method was used to obtain the suitable penalty weightings. First starting with all weightings equal to 1.0. This showed that some criteria differs in magnitude from others. For example the edge length criteria is much larger than balanced edge length one. This means that normalisation
is required. Experiments showed that the weights of the aesthetic criterion presented in Table 6.1 are suitable for this example. After 15 iterations penalty value was reduced from 327.64 to 129.38 (table 6.2).

First figure in Appendix H H.1 displays the initial embedding of Sodor network. After preprocessing function (application of force based algorithm and snapping nodes to grid) is applied the solution looks like Fig. H.2. Solution after 15 iterations is presented in third figure of Appendix H.3. The biggest issue with this solution is that some of its edges violates octilinearity criteria. This is due to algorithm ending up in local minima for nodes of higher degrees (3 or 4).

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balanced edge length</td>
<td>1.00</td>
</tr>
<tr>
<td>Line straightness</td>
<td>0.10</td>
</tr>
<tr>
<td>Octilinearity</td>
<td>10.00</td>
</tr>
<tr>
<td>Angular resolution</td>
<td>2.00</td>
</tr>
<tr>
<td>Edge length</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table 6.1: Weightings for the aesthetic criteria for Sodor network

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>327.640</td>
</tr>
<tr>
<td>1</td>
<td>303.266</td>
</tr>
<tr>
<td>2</td>
<td>296.435</td>
</tr>
<tr>
<td>3</td>
<td>283.055</td>
</tr>
<tr>
<td>4</td>
<td>271.961</td>
</tr>
<tr>
<td>5</td>
<td>265.342</td>
</tr>
<tr>
<td>6</td>
<td>254.482</td>
</tr>
<tr>
<td>7</td>
<td>249.971</td>
</tr>
<tr>
<td>8</td>
<td>231.451</td>
</tr>
<tr>
<td>9</td>
<td>218.448</td>
</tr>
<tr>
<td>10</td>
<td>201.645</td>
</tr>
<tr>
<td>11</td>
<td>189.654</td>
</tr>
<tr>
<td>12</td>
<td>154.374</td>
</tr>
<tr>
<td>13</td>
<td>146.123</td>
</tr>
<tr>
<td>14</td>
<td>130.021</td>
</tr>
<tr>
<td>15</td>
<td>129.380</td>
</tr>
</tbody>
</table>

Table 6.2: Penalty values for Sodor network after each iteration

### 6.2.2 Virgin West Coast network

This network includes 66 locations and 71 route segments connecting them. Similarly as for previous one trial and error method was incorporated to normalise weightings and assign suitable values for
them. This network has more nodes of degree 3 than previous example therefore it is important to sort out correct angles for them. Therefore the angular resolution is assigned larger value. Also some of the locations end up too close to each other after initial embedding. Hence need to emphasise this criteria by setting its weighting to 0.2. Algorithm managed to reduce penalty from 584.976 to 168.132 in 24 iterations. Results are presented in appendix H.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balanced edge length</td>
<td>1.00</td>
</tr>
<tr>
<td>Line straightness</td>
<td>0.10</td>
</tr>
<tr>
<td>Octilinearity</td>
<td>10.00</td>
</tr>
<tr>
<td>Angular resolution</td>
<td>4.00</td>
</tr>
<tr>
<td>Edge length</td>
<td>0.20</td>
</tr>
</tbody>
</table>

Table 6.3: Weightings for the aesthetic criteria for Virgin West Coast network
<table>
<thead>
<tr>
<th>Iteration</th>
<th>Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>584.976</td>
</tr>
<tr>
<td>1</td>
<td>544.353</td>
</tr>
<tr>
<td>2</td>
<td>504.846</td>
</tr>
<tr>
<td>3</td>
<td>481.263</td>
</tr>
<tr>
<td>4</td>
<td>443.946</td>
</tr>
<tr>
<td>5</td>
<td>402.135</td>
</tr>
<tr>
<td>6</td>
<td>381.235</td>
</tr>
<tr>
<td>7</td>
<td>356.783</td>
</tr>
<tr>
<td>8</td>
<td>349.754</td>
</tr>
<tr>
<td>9</td>
<td>325.112</td>
</tr>
<tr>
<td>10</td>
<td>301.743</td>
</tr>
<tr>
<td>11</td>
<td>289.632</td>
</tr>
<tr>
<td>12</td>
<td>273.742</td>
</tr>
<tr>
<td>13</td>
<td>256.935</td>
</tr>
<tr>
<td>14</td>
<td>248.186</td>
</tr>
<tr>
<td>15</td>
<td>245.294</td>
</tr>
<tr>
<td>16</td>
<td>242.746</td>
</tr>
<tr>
<td>17</td>
<td>239.052</td>
</tr>
<tr>
<td>18</td>
<td>235.853</td>
</tr>
<tr>
<td>19</td>
<td>221.748</td>
</tr>
<tr>
<td>20</td>
<td>194.376</td>
</tr>
<tr>
<td>21</td>
<td>189.358</td>
</tr>
<tr>
<td>22</td>
<td>188.895</td>
</tr>
<tr>
<td>23</td>
<td>175.735</td>
</tr>
<tr>
<td>24</td>
<td>168.132</td>
</tr>
</tbody>
</table>

Table 6.4: Penalty values for Virgin West Coast network after each iteration

6.2.3 Conclusion

Visual inspection and analysis of penalty values showed that algorithm performed quite well in producing schematic maps for Sodor and Virgin West Coast networks. They are of comparable quality to solutions produced by Stott [16]. Drawing such maps by hand would be quite challenging. Overall the algorithm performs quite well. However for more complexed networks it does require major tweaks. Especially handling clusters of nodes as oposed to single node at a time.
Bibliography


Appendix A

Personal Reflection

After completion of my second year in School of Computing I decided to do a summer research internship supervised by Dr Raymond Kwan. The goal of my was to produce a prototype system for train crew rostering. Successful completion of the internship and delivery of the goal led me to the industrial placement with Tracsis plc which is the major provider of transport planning software. I really enjoyed my year in industry since working for Tracsis was challenging, interesting and very useful in terms of experience. This led me to choosing final year project with them. Apart from my relationships with the company my choice of project was affected by my personal desire to learn a specific technology (WPF) through the practical work.

The fact that I was doing external project with the particular company strongly affected the process of my project. First of all I took extra care when producing code since the product of my project is likely to be used in practice. This unfortunately left less time for documentation and reporting than originally anticipated.

If I had opportunity to turn back time and manage it more efficiently I would have been less ambitious about the quality of the code and started thinking about the empirical evaluation of the software sooner than I actually did since it took significantly more time than anticipated. This would have provided me with more time to reiterate the report. However I take this as a valuable lesson for the future.

All in all I am happy about the product and my experience in developing it. I believe that Tracsis plc. will make good use of it.
Appendix B

Letters of Support

The following entries in this appendix are:

- Letter of support from John McArthur (CEO of Tracsis plc.)
- Letter of support from Dr Andrew Schwarz (External supervisor, Senior Associate of Tracsis plc)
10 May 2010

To whom it may concern

Mantas Gudynas – Final Year Project

Mantas has worked extremely diligently over the last year and it has been a pleasure working with him. He has applied his theoretical knowledge to practical situations with great skill producing excellent work that has exceeded my expectations. This work will be used as the basis for future enhancements to our TrainTRACS scheduling suite and has genuine commercial value to Tracsis plc.

Working within such a niche market often involves a very steep learning curve and Mantas has approached this with maturity and enthusiasm, acquiring knowledge that can take some people many years to achieve.

I am sure Mantas will prove very successful in any software career he chooses and I would have no hesitation in recommending him. We certainly hope he will come back to Tracsis on graduation and I wish him all the best in the meantime.

John McArthur
CEO, Tracsis plc
10 May 2010

To whom it may concern

Mantas Gudynas – Final Year Project

As a senior developer at Tracsis, I am pleased with the research/work that Mantas has produced during his final year project. The domain of Transport Scheduling is large and complicated and Mantas has done a good job of getting up to speed with the knowledge required to perform the tasks. The prototype application software demonstrated well the visualisation often complex geographic rail data in an easy to understand manner. The addition of functionality allowing the specification of crew depots, depot sub-links and route knowledge is a great improvement over existing table and matrix user interfaces.

On the technical side, I am impressed with the speed at which Mantas has picked up and used Microsoft’s ‘Windows Presentation Foundation’ (WPF) during the short time available on his project. Being advanced topical technologies in the software industry, Mantas’ work adds valuable knowledge and experience to the software development team in Tracsis. From personal experience I know that there are a lot of new concepts to learn and that WPF often requires a different approach and way of thinking compared to conventional user interface design.

The software engineering principles and platform adopted are aligned with the high standards Tracsis is following. Mantas’ use of design patterns, particularly his use of the Model-View-View Model (MVVM) pattern have resulted in code that is loosely coupled and easily extendable.

Overall it has been a pleasure being involved in Mantas’ final year project and I wish him luck with his future in software.

A. Schwarz

Dr Andrew Schwarz
Senior Associate, Tracsis plc
Appendix C

Gantt Charts
### Initial Project Plan

<table>
<thead>
<tr>
<th>Activity</th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Week 8</th>
<th>Week 9</th>
<th>Week 10</th>
<th>Week 11</th>
<th>Week 12</th>
<th>Week 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Researching relevant literature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Designing initial architecture model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Producing initial user interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Producing data model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementing the algorithm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Revising architecture model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Designing and implementing usability evaluation framework</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performing algorithm tests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writing the report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Revised Project Plan

<table>
<thead>
<tr>
<th>Activity</th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Week 8</th>
<th>Week 9</th>
<th>Week 10</th>
<th>Week 11</th>
<th>Week 12</th>
<th>Week 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Researching relevant literature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysing data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Producing data model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Designing initial architecture model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Producing initial user interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementing the algorithm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Revising architecture model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementing changes to user interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Designing and implementing usability evaluation framework</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performing algorithm tests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writing the report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix D

Empirical Evaluation Test
Appendix E

Custom Implementation of UndoableCommand

```csharp
public class RemoveDisconnectedLocationsCommand : RelayCommand, IUndoableCommand {

    // Private fields
    private MapViewModel _mapViewModel;
    private IEnumerable<LocationViewModel> _removedLocationViewModels;

    // Constructor of the class
    public RemoveDisconnectedLocationsCommand(MapViewModel mapViewModel)
    {
        this._mapViewModel = mapViewModel;
        base.Execute1 = new Action(ExecuteCommand);
    }

    // Main command method to be used as an action
    private void ExecuteCommand()
    {
        _removedLocationViewModels = _mapViewModel.LocationViewModels
            .Where(lvm => lvm.Location.Legs.Count() == 0).ToList();
    }
}
```
foreach (LocationViewModel lvm in _removedLocationViewModels)
    _mapViewModel.MapObjects.Remove(lvm);

UndoRedoContext.Instance.AddCommand(this);
}

public void Redo()
{
    foreach (LocationViewModel lvm in _removedLocationViewModels)
        _mapViewModel.MapObjects.Remove(lvm);
}

public void Undo()
{
    foreach (LocationViewModel lvm in _removedLocationViewModels)
        _mapViewModel.MapObjects.Add(lvm);
}
Appendix F

Screenshot of the Software
Appendix G

Empirical Evaluation Results

Figure G.1: Evaluation test completion times
Figure G.2: Number of clicks to complete the evaluation test

Figure G.3: Mistakes made in evaluation test
Appendix H

Algorithm Solutions

H.1 Sodor Network

Figure H.1: Initial solution for Sodor network
Figure H.2: Sodor network after force and snap functions applied

Figure H.3: Solution of Sodor network
H.2 Virgin West Coast Network

Figure H.4: Initial solution for Virgin West Coast network

Figure H.5: Virgin West Coast network after force and snap functions applied
Figure H.6: Solution for Virgin West Coast Network after 24 iterations