Design Specification

jok13

’

CONTENTS

GLOSSARY

- **PRM**
  Person with Reduced Mobility
- **AU**
  Aberystwyth University
- **the system**
  Access Aber - A routeplanning-application aimed at PRMs on the AU campuses
- **Node**
  a single coordinate
- **Relation**
  Descriptive collection of nodes
- **OSM**
  OpenStreetMap
- **O()**
  Big-O notation. Used to illustrate the worst-case time/memory requirement-growth of the system with respect to some constant **n**
- **In-place**
  Refers to an algorithm able to transform its input using no auxiliary data structure.
  Usually requires $O(log\, n)$ to $O(n)$ extra space
- **Complete**
  The system is able to find a path to any location, provided that it is possible to find one
- **Optimal**
  The route found by the system is always the shortest/best

I. Introduction

A. Purpose of this document

This document describes the outline design for the system I am developing as part of my Major Project - *Access Aber*. This project will form the basis of my dissertation written as part of my MSc at Aberystwyth University.

B. Scope

This design specification will break down the coding-components of my system, and detail how they are connected, what they do, and what the end-result of the system should look like.

C. Objectives

- Describe the main components of the system
- Detail the dependencies between the different parts of the system
- Provide a reference to any design-models, design-patters and tools that were used to develop the system

II. System Overview

A. Data-sources

The data used to plan routes in this system will most likely come from an external source, as the mapping of roads, stairs, buildings, etc. is a very time-consuming task to undertake, and many external sources are fairly up-to-date with the layout of the AU campuses.

1) Data-filtering: The system is aimed at PRMs, so any routes it returns to the user has to be accessible.

Unless the external data has already been tailored to a specific disability when it is passed to the system: some method of flagging and/or deletion of nodes and relations will be needed.

2) Versatility: The data used in my system is (currently) from OSM, but the system is designed to work with data from any external source - as long as it is in XML-format, all nodes and relations are individually marked with proper XML-tags, and all the required fields like ID, latitude, longitude, etc. are present.

3) Output: The output of the system is also designed to be quite versatile, as it consists solely of an ordered list of coordinates - ready to be plotted on any map (as long as it uses the same coordinate-system or alternatively can convert the coordinates). The coordinate-system used in the output could also be converted beforehand if the coordinate-system used by the map is not the same as the input to the system.

B. Time and Space-constraints

As the system is meant to plan routes for pedestrians, it needs to run within some acceptable parameters.

1) Time: Any route-planning application should be able to display routes to its users in real-time, therefore it needs to have a very good time-complexity: preferably in the $O(1)$ to $O(log\, n)$ region.

2) Memory: Any application running on a handheld device should manage its memory-usage quite carefully, as the storage-capacity on these devices can be quite limited.

My system requires the input of an initial database of nodes and relations, but should ideally not need to store much more than this initial overhead. It will of course require some additional memory, but the usage should never get to the point where it suddenly reaches exponential growth. Things like the reuse of data in the form of Pointers (Or ‘References’in Java), and implementing algorithms able to work in-place, may be useful to consider.

III. Design Considerations

A. Assumptions and Dependencies

- The user is assumed to be a pedestrian
- The input can be dynamic when passed to the system, but static once loaded
- The routes are planned dynamically
- The pathfinding is complete and optimal
- The input-data is assumed to contain the following:
  - Only one input-file
  - The input-file is formatted as an XML-document
  - Nodes and relations can be clearly distinguished by their XML-tags - the ordering in which they appear is irrelevant
  - Nodes and relations are only listed once:
    - Unique IDs within the set of nodes.
    - Unique IDs within the set of Relations.
  - All nodes have a latitude and longitude (coordinate)
  - Relations are labelled correctly and accurately - describing what they represent (e.g. stairs or a footway)
  - Nodes are ordered by their actual physical order within each relation.
  - A node can be part of multiple relations.
- All coordinates in the input use the same coordinate-system
- The output is directly related to the original, unmodified, state of the input
- The system is able to handle input in any coordinate-system (assuming the coordinate-system is constant)

B. General Constraints

- Everything is written in Java.
  This limits the number of libraries I can use, and which platforms the system can be deployed on.
- limited memory available
  Memory-use has to be managed very carefully, and can never be allowed to grow out of control.
  This may have a negative impact on the run-time of the system.
- User assumes real-time functionality
  A route has to be returned to the user almost as soon as they input where they want to go. Any worse time-complexity than this, and the user might stop using the system.
  This may have a negative impact on memory-usage
- The pathfinding relies upon accurate and updated data
  The data used by the system has to either be downloaded from the internet, or stored locally and updated regularly in order to minimise routing-errors.
- The OSM data is licensed under the 'Open Data Commons Open Database License' **(ODbL)**
  The OSM data can be stored and edited by the system, but it must always abide by this license
- OSM's map-cartography is licensed under the 'Creative Commons Attribution-ShareAlike 2.0 license' **(CC BY-SA)**
  We have to abide by this license
- I plan on using Test Driven Development
  This means that every class and method in the system has to be rigorously tested. Any deviation from this may result in unforeseen issues when maintaining the code later.
  These tests also act as a form of self-maintaining design-documentation - clearly showing what functionality we assume to be present

C. Goals and Guidelines

- Emphasis on lowering both speed and memory use.
  Neither is really more important than the other, but I have found that improving one often leads to improvement in the other as well.
  Care has to be taken to not ignore one when optimising the other though.
-

D. Development Methods

1) Test Driven Development: Test Driven Development makes sure that all functionality is preserved in the system whenever parts if it is restructured.

By creating unit tests for every class and method in the system, it becomes much easier and quicker to discover bugs and dangerous behaviour after refactoring the code.

But even though all of the unit tests pass, the system might still have some bugs in it that we didn't think of when writing the tests. So TDD is really just a tool to make it easier to refactor code, and for documenting what behaviour we expect in each of the individual classes and methods.

2) Spiral Model: The spiral model makes sure that the development-process is a little more structured.

The model follows these four steps:

1) determine objectives, alternatives, and constraints of the iteration
2) evaluate alternatives; identify and resolve risks
3) develop and verify deliverables from the iteration
4) plan the next iteration

The spiral model makes it easier to focus on one aspect of the system at a time, and makes sure that that aspect is implemented properly. It does this by having a step dedicated to finding and evaluating alternative approaches based on the risk involved in implementing them. This risk could be: time vs memory requirements, expected difficulty of coding, etc.

After one iteration of the spiral is complete, we are able to move on to another aspect of the system. This makes it much easier to stop optimising every single part of the system, and instead prioritise creating a finished product.

IV. Architectural Strategies

A. Strategy-1 name or description

B. strategy-2 name or description

C. ...

V. System Architecture

A. component-1 name or description

B. component-2 name or description

C. ...

VI. Policies and Tactics

A. policy/tactic-1 name or description

B. policy/tactic-2 name or description

C. ...

VII. Detailed System Design

A. module-1 name or description

B. module-2 name or description

C. ...

References

[1] “title.”