|  |
| --- |
| Mission Solutions engineering |
| Data-Driven Decisions Aid Tool |
| Detailed Design Document |
|  |
| **Shahid Akhter, Kevin Friesen, Stacey Montresor, Matthew Mullen, Jonathan Summerton** |
|  |

|  |
| --- |
| Course: Software Engineering I, Fall 2010 |
| Instructor: Dr. Adrian Rusu |

Table of Contents

[1. Introduction 3](#_Toc277012785)

[2. Architectural Overview 4](#_Toc277012786)

[3. Input Module 5](#_Toc277012787)

[4. Path Module 8](#_Toc277012788)

[5. Output Module 11](#_Toc277012789)

[6. Abstract Data Types 12](#_Toc277012790)

[7. Algorithm Design 13](#_Toc277012791)

[8. COMPRISES Diagram 14](#_Toc277012792)

[9. USES Diagram 15](#_Toc277012793)

[10. Integration Test Plan 16](#_Toc277012794)

# Introduction

This is the official detailed design document, per the requirements specified in the SRS, for the MSE Data-Driven Decision Aid Tool

**Input**

**User Input**

* Start and end Location in longitude and latitude
* Preferred parameters including: safest, fastest, most direct or any combination of the three

**Event Data**

* Data collected on area within scope of possible paths

**Path Module**

**Weighting**

* Assigns severity to edges based on parameters selected

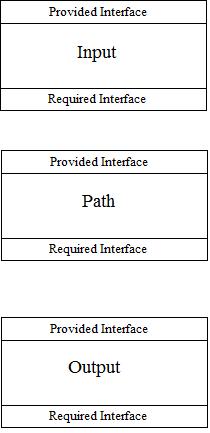
**Path Finding**

* Provides top three paths from path ranking module

**Output**

* Displays top three paths color coded green (best), yellow (second best) and red (third best)

# Architectural Overview



**Figure 2.1 High Level Architecture Overview**

One of the key components in the Data-Driven Decisions Aid Tool is the Path module. This module will accomplish two important tasks. The Path Weighting module weights the edges of the map based on the selected parameters. Then the Path Finding module determines the best three paths based on the same parameters.

The Data-Driven Decisions Aid Tool will be driven by the Input module. This module will provide the user input and the current event data to the Path module. Upon completion the Path module will pass the found paths to the Output to be displayed for the user.

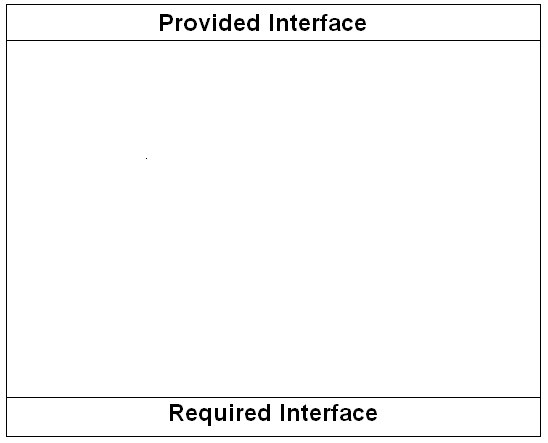
The rest of the detailed design document is as follows. Sections three through five describe each module in detail. Section six covers all of the abstract data types that will be used by modules in this design. Section seven defines the algorithms that will be used in determining paths. Section eight and nine include the COMPRISES and USES diagrams respectively. Lastly section ten contains the integration test plan.

# Input Module

**Purpose**: The purpose of this module is to receive input parameters from the user and event data from a KML file that will help in the process of finding the correct path(s) for the user to follow. The module takes all this information and then passes it on to the Path module.

**Rationale:** This module is used to receive input from the user, and to read in data from a KML file that contains road information, and event data.

**High Level Module Design**: The input module is broken down into three sub-modules. These modules are User Parameter, Event Data, and KML Parser sub-modules. These sub-modules are shown in the figure 3.1.



Provided Interface

User Parameter

Required Interface

Event Data

Required Interface

**Input**

Provided Interface

Event Data

Figure 3.1 Input Module Overview

Provided Interface

KML Parser

Required Interface

**Provided Interface:**

* This module has no provided interface.

**Required Interface:**

* The union of required interfaces of User Parameter and KML Parser sub-modules.

**3.1 User Parameter Module**

**Purpose:** The purpose of this sub-module is to collect the information that the user enters as input consisting of start Location, end Location, and type(s) by which to filter the paths the user wishes to receive as output.

**Rationale:** The user needs to enter valid starting and ending Locations and select desired type(s) in order for the correct paths to be returned by the Path module.

**Provided Interface:**

* This module has no provided interface.

**Required Interface:**

* Path Weighting
* void selectedParameter(Location start, Location end,

boolean safest, boolean fastest, boolean shortest)

A complete definition of the functionality of this required interface can be found in the Path Weighting module’s provided interface.

**3.2 Event Data Module**

**Purpose:** The purpose of this module is to store the KML file containing the current event data for the area in which the user will pick a path through.

**Rationale:** The information about the related area will be used to set the severity weights used in determining the correct paths based on selected parameters.

**Provided Interface:**

* This module has no provided interface.

**Required Interface:**

* KML Parser
* void currentEventData(KML file)

A complete definition of the functionality of this required interface can be found in the KML Parser module’s provided interface.

**3.3 KML Parser Module**

**Purpose:** The purpose of this module is to convert the event data in the form of a KML file into a graph of the area with vertices and edges. This graph will be used in determining the best paths.

**Rationale:** In order for the Path Finding module’s algorithm to determine the correct paths, the KML file needs to be converted into a graph of vertices and edges.

**Provided Interface:**

* void currentEventData(KML file)
* **Description:** This function will convert the KML file containing the event data for the desired area from the Event Data module into a graph of vertices and edges.

**Required Interface:**

* Path Weighting
* void parsedData(graph)

A complete definition of the functionality of this required interface can be found in the Path Weighting module’s provided interface.

# Path Module

**Purpose:** The purpose of the Path module is to find the best three routes between two given points, based on parameters selected by the user.

**Rationale:** This module contains all the information and functions needed to find the best three routes between two points based on parameters selected by the user.

**High Level Module Design:** The Path module is broken down into two sub-modules. These modules are Weighting and Path Finding. These sub-modules are shown in figure 4.1.

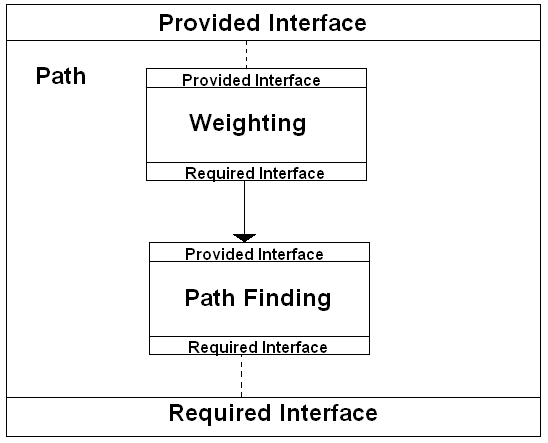


Figure 4.1 Path Module Overview

**Provided Interface:**

* + The Weighting module’s provided interface.

**Required Interface:**

* + The Path Finding module’s required Interface.

**4.1 Weighting Module**

**Purpose:** The purpose of this module is to set the weight of each road segment, so that Djkstra's algorithm can be performed on the graph that correlates to the roads and intersections of the area. Given the input parameters (fastest, safest, and/or shortest) this module will calculate the rating (a number in the range from 0 to 1) which will serve as the weight of the edge on the graph. When multiple parameters are used, then the ratings of the given parameters will be multiplied together, in order to keep the number in the range from 0 to 1.

**Rationale:** The reason this module is separate from the path finding module, is that the process of setting weights for all edges on the graph is completely different than the process of finding a path through that graph. It is also assumed that this module will have to be run before generating a path, or else none of the edges on the graph will have a weight, hence any path generated will give no useful information.

**Provided Interface:**

* + void parsedData(graph)
    - **Description:** This function will receive the graph constructed from the KML parser.
  + void selectedParameter(Location start, Location end,

boolean safest, boolean fastest, boolean shortest)

* **Description:** This function will use the booleans to determine the proper weights to set, and the Coordinates to set the start and end of the paths.

**Required Interface:**

* + Path Finding
    - void weightedGraph(graph)

A complete definition of the functionality of this required interface can be found in the Path Finding module’s provided interface.

**4.2 Path Finding Module**

**Purpose:** The main goal of this path finding module is to find the top three paths between point A and point B based upon parameters which emphasize safeness, quickness, directness, or a combination of these parameters.

**Rationale:** This module utilizes Djkstra’s algorithm to determine the top three paths based on selected parameters.

**Provided Interface:**

* + void weightedGraph(graph)
    - **Description:** This function receives the weighted graph from the Weighting module.

**Required Interface:**

* + Output
    - void topRoutes(List<Routes>)

A complete definition of the functionality of this required interface can be found in the Output module’s provided interface.

# Output Module

**Purpose:** The output module’s purpose is to present the top three paths in a visual format. The top three paths will be displayed color coded. The best path will be displayed in green. The second best path will be displayed in yellow, with the third best path displayed in red.

**Rationale:** Displaying the paths color coded gives the user a visual that allows the user to easily distinguish one path from the other two.

**Provided Interface:**

* + void topRoutes(List<Routes>)
    - Receives the collections of Locations representing the best three paths found.

**Required Interface:**

* + This module has no required interfaces.

# Abstract Data Types

**Coordinates:** Represents latitude and longitude points of particular locations.

**Event Data:** Statistics are reported via KML files that are centralized on specific Coordinates. The statistics include:

* + Date the event occurred
  + Time the event occurred in 24 hour or military representation
  + Severity rating range from 0 to 10
  + Impact of the event
  + Shape (area effected by the event):
    - OriginPoint: Location
    - Type
    - Points[]: Location
    - Dimensions:
      * Name
      * Value
  + Location:
    - Latitude
    - Longitude
    - Altitude

**Graphs:** The conversion result of a KML file.

* Vertex represents an intersection
* Edge represents a road segment
  + Weight is assigned based on parameters and event data

**Vehicle Object:** Stores statistics on vehicle being used for travel.

* Maximum vehicle speed
* Width of vehicle
* Turning radius

**Road Data:** Statistics related to road segments.

* Intersections (Location) comprising road
* Speed of road
* Width of road

**Route:** Collection of Locations from point A to point B.

# Algorithm Design

The Data-Driven Decision Aid Tool uses two algorithms in determining the top three paths. The first algorithm is the weighting algorithm. The second algorithm is the path finding algorithm.

The weighting algorithm assigns weights to every edge of the graph based on selected parameters. The procedure to determine the weights based on the three parameters vary. The following is detailed descriptions of each:

* Safest: This weight is stored in the event data as severity. It will be converted to be within the range from 0 to 1, with one being the most severe.
* Fastest: The weights for these edges will be determined by the average road speed for the equivalent segment over 60 minutes times the length of the segment.
* Shortest: The weights for these edges will be determined by the length of the segements.

The path finding algorithm is an implementation of Djkstra’s algorithm. After the weights are placed on the graph, Djkstra’s is called to determine the best path. After the first path is determined, the weight on this path is adjusted so that it will not be considered the best. Djkstra’s is then called again. This time it will return the second best path. Like the first path the second path’s weight is altered and Djkstra’s is called one last time, returning the third best path.

# COMPRISES Diagram

Input

Output

Path Finding

Weighting

Path

KML Parser

Event Data

User Parameter

# USES Diagram

(0)

(0)

Provided Interface

User Parameter

Required Interface

Provided Interface

Event Data

Required Interface

(4)

(3)

(2)

(1)

Provided Interface

KML Parser

Required Interface

Provided Interface

Weighting

Required Interface

Provided Interface

Path Finding

Required Interface

Provided Interface

Output

Required Interface

# Integration Test Plan

*(to be completed by Fairfield)*