Navigation System v3

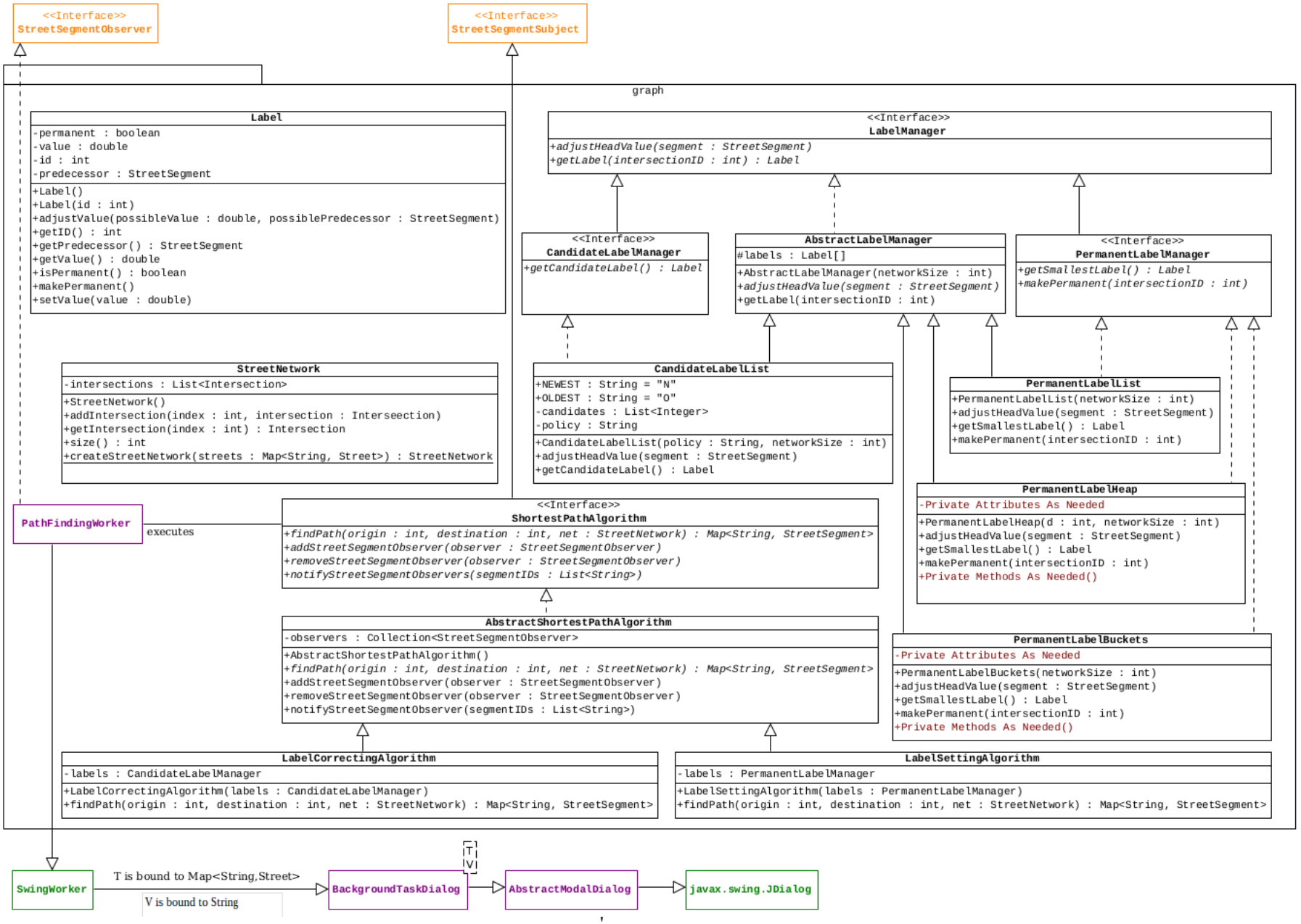
Purpose

Version 3 of the navigation system will provide the user with the ability to calculate shortest paths.

Design

The design of the system is summarized in the following UML class diagrams. Note that the components in jade green are part of the Java API, the components in purple have been provided to you, and the components in orange are "old".

The Future of Navigation



Specifications

This section contains design specifications for some of the components above. For the others, the UML diagrams should provide all of the information that you need.

The `Label` Class

`Label` objects are used in label setting and label correcting algorithms. They maintain information about the shortest path to a particular `Intersection` that has been found thus far, including the length of the path and the incoming `StreetSegment` (i.e., the `StreetSegment` from the predecessor `Intersection`) on that path.

The `adjustValue()` method must only update the `value` and `predecessor` if the `possibleValue` is less than the current `value`.

Note that the `isPermanent()` and `makePermanent()` methods will only be used in label setting algorithms.

Classes that Realize the `LabelManager` Interface

The `adjustHeadValue()` method in classes that realize the `LabelManager` interface must adjust the `Label` at the head node of the given `StreetSegment`. It must invoke the `adjustValue()` method of the appropriate `Label` object so that the `Label` is only updated if its value would be reduced.

Classes that Realize the `CandidateLabelManager` Interface

The `getCandidateLabel()` method in classes that realize the `CandidateLabelManager` interface must return an appropriate candidate label. There are many ways to accomplish this; no particular algorithm has been specified.

Classes that Realize the PermnanentLabelManager Interface

The `getSmallestLabel()` method in classes that realize the `PermanentLabelManager` interface must return a `Label` that has the minimum value among all non-permanent `Label` objects. A `PermanentLabelList` object must search through all non-permanent `Label` object in the `List`, a `PermanentLabelHeap` object must use a *d*-heap for this purpose, and a `PermanentLabelBuckets` object must use buckets for this purpose.

Classes that Extend the AbstractShortestPathAlgorithm Class

Classes that extend the `AbstractShortestPathAlgorithm` class must use an appropriate `LabelManager` to manager the labels that are used in the `findPath()` method.

The `StreetSegment` objects in the shortest path must be returned by the `findPath()` method.

The `findPath()` method **may** (but is not required to) inform `StreetSegmentObserver` objects of `StreetSegment` objects that it has identified while performing the calculations. This functionality will not be used in the final product, but is useful when debugging and profililng.

Examples

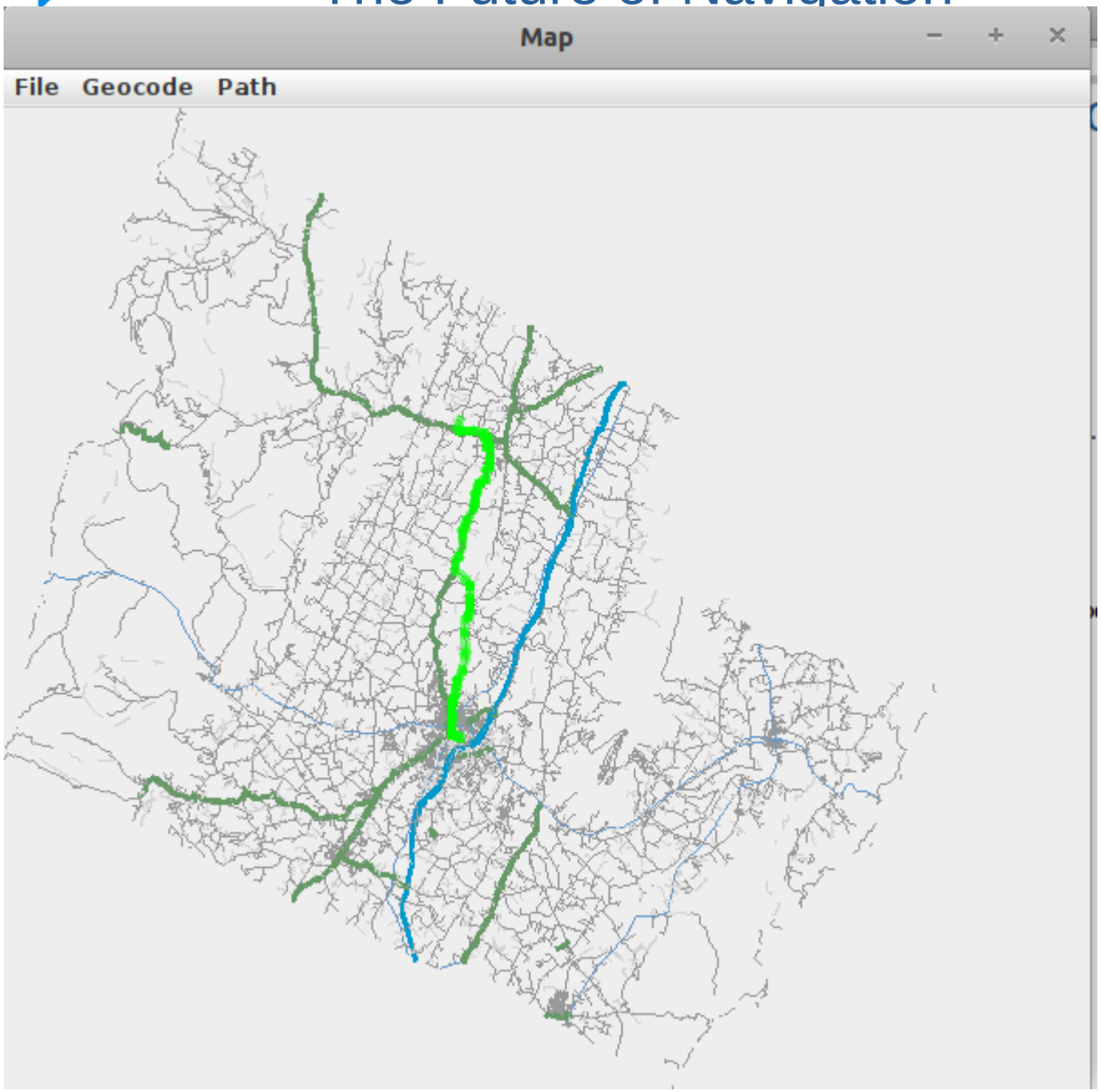
This section contains examples of what we hope the maps will look like.

The first shows the streets in and around Rockingham County, VA with the shortest path from 400 Paul St to 7000 Jess & Mary Ln.

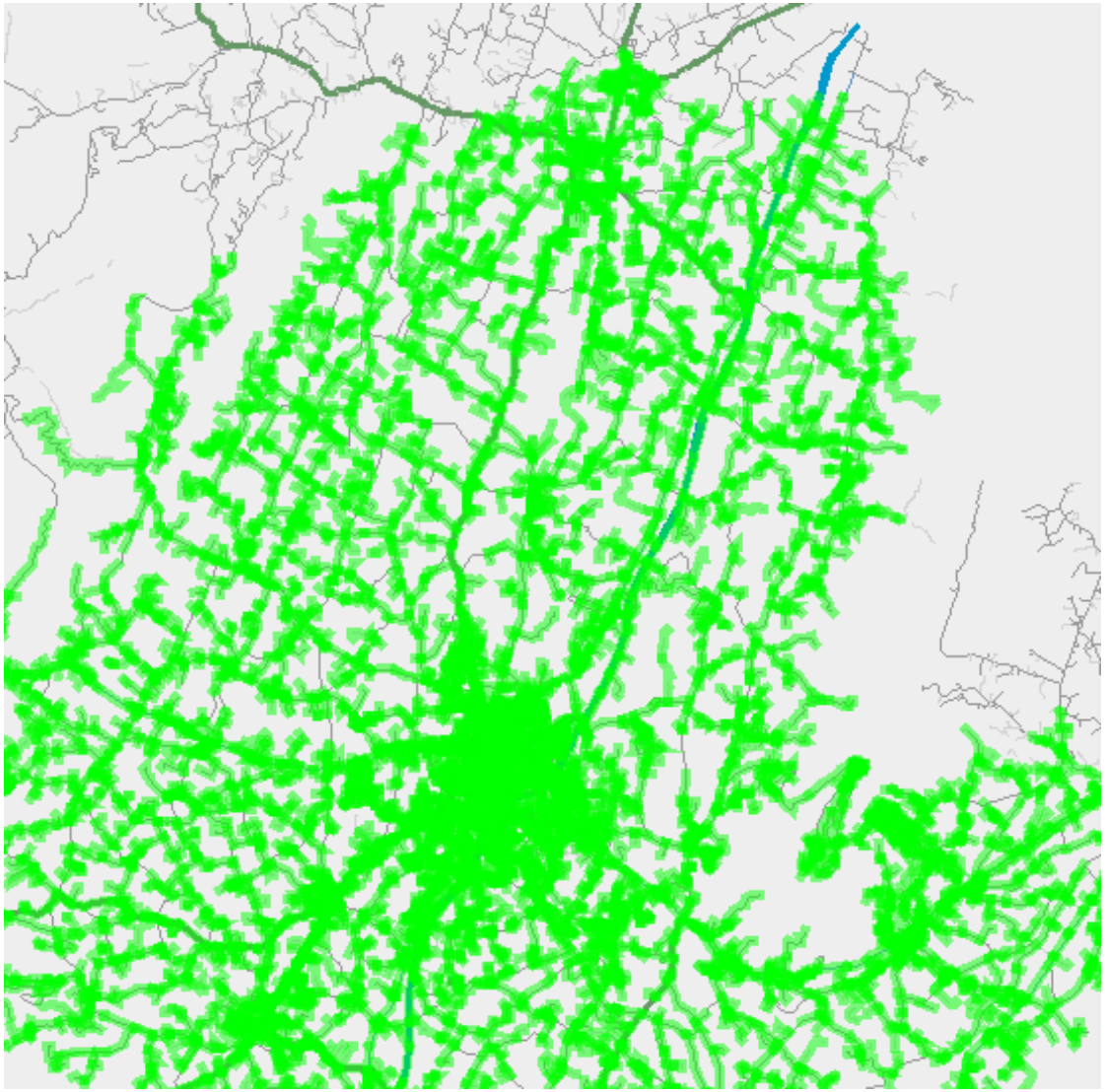
The second shows part of the "shortest path tree" (i.e., the intermediate calculations) calculated using a label correcting algorithm for the same origin and destination.

The third shows the streets in Virginia, with the shortest path from 400 Paul St (with map coordinates in kilometers of about (1469, 239) to 140 Arlington Ave (with map coordinates in kilometers of about (1217, -11)).

The Future of Navigation



The Future of Navigation



The Future of Navigation