Oregon State University | College of Engineering

AN OPEN SOURCE TOOL FOR THE VISUALIZATION, ANALYSIS AND REPORTING OF OREGON'S REGIONAL AND STATEWIDE TRANSIT NETWORKS

Project Number SPR-13-075

FINAL PROJECT REPORT

Prepared for
Oregon Department of Transportation
Research Unit
200 Hawthorne Ave. SE, Suite B-240
Salem, Oregon 97301-5192

Revision Date: 1/8/2015

Prepared by:

J. David Porter, David S. Kim, Saeed Ghanbartehrani, Alireza Mohseni
School of Mechanical, Industrial and Manufacturing Engineering
Oregon State University

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

This document constitutes the final report for the Oregon Department of Transportation (ODOT) research project titled "*An Open Source Tool for the Visualization, Analysis, and Reporting of Regional and Statewide Transit Networks*". This project is a continuation of the ODOT research project "*Proof of Concept: GTFS Data as a Basis for Optimization of Oregon's Regional and Statewide Transit Networks*" *(1)* where the initial prototype of the Transit Network Analysis (TNA) software tool was developed.

The objective of this project was to enhance the reporting and interface features of the TNA software tool so that it may provide more value to planners and analysts. Some major reporting enhancements are the generation of reports by pre-defined and custom specified geographic areas, and the inclusion of U.S. Census Bureau data in the reports. To accommodate these and future anticipated enhancements, a major part of this project was to change the underlying data structure of the TNA software tool.

The initial prototype of the TNA software tool completed in December 2013, was developed using the OpenTripPlanner (OTP) framework. OTP is a multi-modal trip planner launched in 2009 as an open source, community driven trip planner that utilizes GTFS data and provides visualization capabilities. The underlying data structure used in OTP is referred to as a "graph" file, which is a text file implementing XML encoding of data. This data structure works well with OTP where the emphasis is individual trip planning rather than system report generation. Because the new reporting enhancements of the TNA software tool require large numbers of queries, which can be specific to non-continuous geographic areas, a PostgreSQL relational database was developed as the underlying TNA software tool data structure. PostgreSQL has features that make it particularly applicable to efficient and extendable reporting capabilities where geographic information is involved. While it may have been possible to utilize the OTP graph file data structure, such a platform is not built to process geographic-based queries and lacks adequate documentation for developers.

After developing and populating the PostgreSQL database, the project focused on implementing various reporting, interface, and administrative capabilities. These enhancements are described in this report.

1.1 REPORT AUDIENCE AND OBJECTIVES

This report in written for two main audiences. The first audience is the ODOT project sponsors with the objective of providing a comprehensive overview the TNA software tool architecture and the functional enhancements completed. The second audience is the community of open source developers. For these developers the goal of this report is to provide sufficient documentation so that when used in combination with documented source code, and available open source framework component documentation the development of TNA software tool extensions is accessible.

1.2 REPORT ORGANIZATION

The remainder of this report is organized as follows. Chapter 2 describes the TNA software tool software architecture, and also the software framework components and development tools utilized. Chapter 3 presents the details of the relational database developed as the data repository for the TNA software tool. Chapter 4 describes the enhancements implemented and is organized into server-side and client-side development tasks. Chapter 5 presents conclusions and recommendations for future work.

2.0 DEVELOPMENT OF THE TNA SOFTWARE TOOL

In this chapter, the development work conducted to produce the *transit network analysis* (TNA) software tool is described.

The rest of this chapter is organized as follows. Section 2.1 describes the software architecture and the main modules of the TNA software tool, whereas section 2.2 describes the software tools and software libraries used to add functionality to the TNA software tool.

2.1 SOFTWARE ARCHITECTURE AND MAIN MODULES

Figure 2.1 depicts the high level software architecture and the main modules of the TNA software tool. As Figure 2.1 shows, the TNA software tool utilizes a client-server architecture.

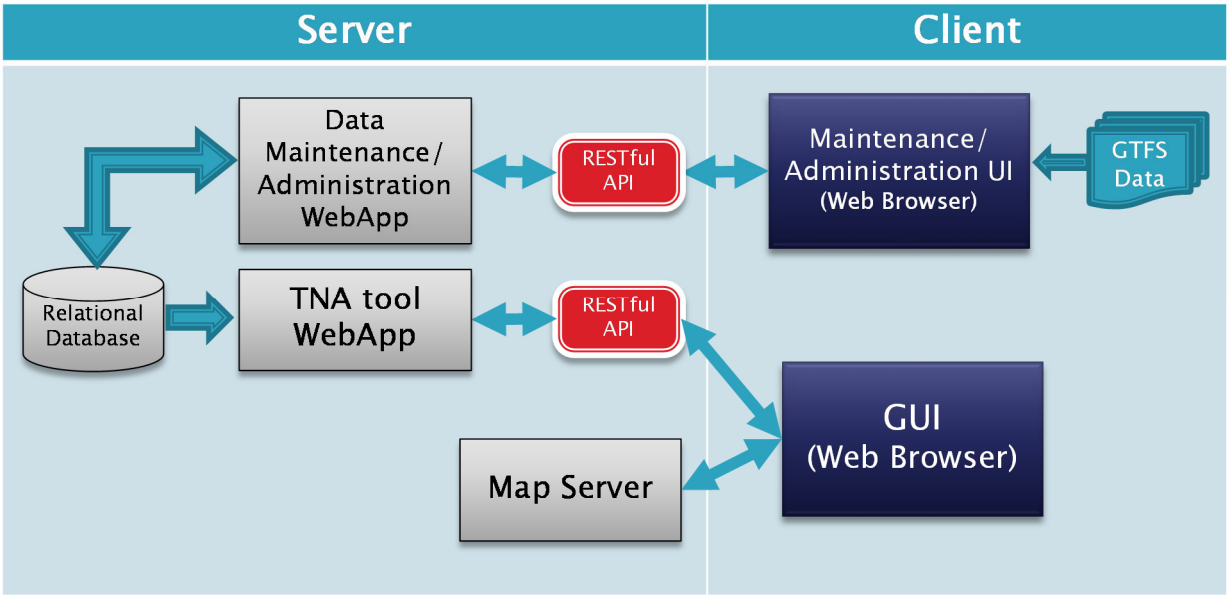


Figure 2.1: Software architecture and main components of the TNA software tool

The resource intensive computations are performed by the server side of the software architecture, thus allowing the TNA software tool to run on any operating system (i.e., Microsoft® Windows, Mac OS, and Linux) and hardware configuration. The TNA software tool uses a relational database as the basis for data storage and retrieval. The module "Data Maintenance/Administration WebApp" is used to load GTFS data into the relational database through the "Maintenance/Administration UI" module using a web browser. The module "TNA tool WebApp" is responsible for processing requests from the Graphical User Interface (GUI), extracting the requested data from the relational database, creating the response (e.g., a report, the location of transit stops, etc.), and sending it back to the GUI. A map server can be used to provide map tiles for the geographic area being displayed on the GUI. The "RESTful API" is the main interface between the GUI and the web applications on the client side of the software

architecture. The majority of the server side modules of the software architecture were developed with the Java programming language.

On the client side, the only requirement is a web browser which acts as the GUI. The TNA software tool is optimized to run on the Mozilla Firefox and the Google Chrome web browsers.

2.2 ADDITIONAL SOFTWARE DEVELOPMENT TOOLS

In this section, the software tools and software libraries used to add functionality to the TNA software tool are introduced.

Eclipse was chosen as the integrated development environment (IDE) for developing the TNA software tool. Eclipse is a free and open source IDE which can be used to develop applications in the Java programming language, as well as other programming languages such as C, C++, JavaScript, Python, and CSS, to name a few *(2)*. Additional plug-ins such as EGit, m2eclipse, and WTP, were installed in Eclipse to enable all the required functionality to develop the TNA software tool.

The main module on the server side of the software architecture is an open source PostgreSQL database engine. The PostgreSQL database engine was supplemented with PostGIS to support spatial data types (e.g., point, line, polyline, etc.) and spatial functions and queries. The additional libraries used in the development of the server side modules are listed in Table 2.1.

Table 2.1: Server Side Libraries

Name	Description	Reference
`Hibernate ORM`	Hibernate is an open source object-relational mapping library mapping for Java.	*(3)*
`Hibernate Spatial`	Hibernate spatial is an open source object-relational mapping library with support for spatial databases.	*(4)*
`onebusaway-gtfs`	An open source library in Java with classes for GTFS entities.	*(5)*
`onebusaway-gtfs-hibernate`	An open source library in Java that uses hibernate to query GTFS data from relational databases.	*(6)*
`onebusaway-gtfs-hibernate-cli`	An open source library that uses hibernate for reading and loading GTFS data into relational databases.	*(7)*
`GeoTools`	An open source Java library that provides tools for processing geospatial data.	*(8)*

The client side functionality of the TNA software tool was implemented using the hypertext markup language (HTML) *(9)*. Cascading Style Sheets (CSS) *(10)* was the tool used for styling the web pages and JavaScript *(11)* was the main scripting language used in the GUI.

Table 2.2 provides a brief description of the additional JavaScript plug-ins and libraries that were used to add functionality to the client side of the TNA software tool.

Table 2.2: Client Side Libraries

Name	Description	Reference
jQuery	jQuery is a feature rich JavaScript library that is compatible with most web browsers.	*(12)*
jQuery UI	jQuery UI is a plug-in for jQury that provides some effects, widgets and interactions for jQuery.	*(13)*
jQuery UI DataTables	A JavaScript library used for displaying reports in tables with features like searching, sorting, and exporting.	*(14)*
jQuery UI dialogextend	An open source JavaScript library that extends features of the standard jQuery UI dialog box.	*(15)*
Bootstrap Dropdown	An open source JavaScript library for generating dropdown menus.	*(16)*
jstree	An open source free JavaScript plug-in that provides a tree menu structure.	*(17)*
Leaflet	Leaflet is a versatile and widely used JavaScript library, which is open source and free. It is used for building interactive maps.	*(18)*
Leaflet markercluster	Leaflet markercluster is a JavaScript library that provides clustering capability for leaflet markers. Markercluster can display up to 50,000 markers on map efficiently.	*(19)*
Leaflet.encoded	An open source leaflet library used to decode the encoded polyline into an array of L.LatLng objects that can be displayed on a map.	*(20)*
Leaflet MiniMap	An open source leaflet library that adds mini map feature to LeafLet maps.	*(21)*
Google Maps API	A free library that allows using Google maps features such as map tiles and street view.	*(22)*
OpenStreetMap API	A free library for using OpenStreetMap features like location search.	*(23)*

3.0 DATABASE DESIGN AND IMPLEMENTATION

This section discusses the development efforts to implement the PostgreSQL relational database that is the basis of the TNA software tool.

3.1 ACQUIRING UP-TO-DATE OREGON SHAPE FILES AND CENSUS DATA

Shape files for Oregon counties, congressional districts, census tracts, census places, urbanized areas, and census blocks were downloaded from the website of the US Census Bureau *(24)*. The TNA software tool uses population data at the census block level to calculate population statistics around stops. The population data used by the TNA software tool corresponds to the 2010 census conducted by the US Census Bureau.

The shape file data are imported into the PostgreSQL relational database using the shape file import wizard. This shape file import wizard is simple to use and loads information from the shape files into the tables in the relational database that have the same name as the input file. Population and other metadata (which comes with the census data in a separate DBF file) are also imported to the PostgreSQL database.

3.2 ACQUIRING GTFS DATA

The most up-to-date GTFS data for Oregon transit agencies were retrieved from the web site oregon-gtfs.com as a single zip archive. As described in section 2.2.1, the archive was unzipped into a local folder on the server and GTFS feeds for each individual transit agency were loaded one by one into the PostgreSQL relational database through the web browser.

3.3 DATABASE SCHEMA

The Entity-Relationship (ER) diagram that represents the schema of the PostgreSQL relational database is depicted in Figure 3.1. The tables included in the schema can be classified into four different categories:

- **GTFS data tables**. These tables are created and populated by the module "onebusaway-gtfs-hibernate-cli".
- **GTFS data lookup tables**. These tables are created manually (based on GTFS data tables) by running PostgreSQL queries to speed up more complex queries.
- **Census data tables**. These tables contain census data and are created by importing shape files into the PostgreSQL relational database using the shape file import wizard.
- **Census data lookup tables**. These tables are created manually based on census data tables by running PostgreSQL queries to speed up more complex queries.

In the following sections, the tables in each category are further explained.

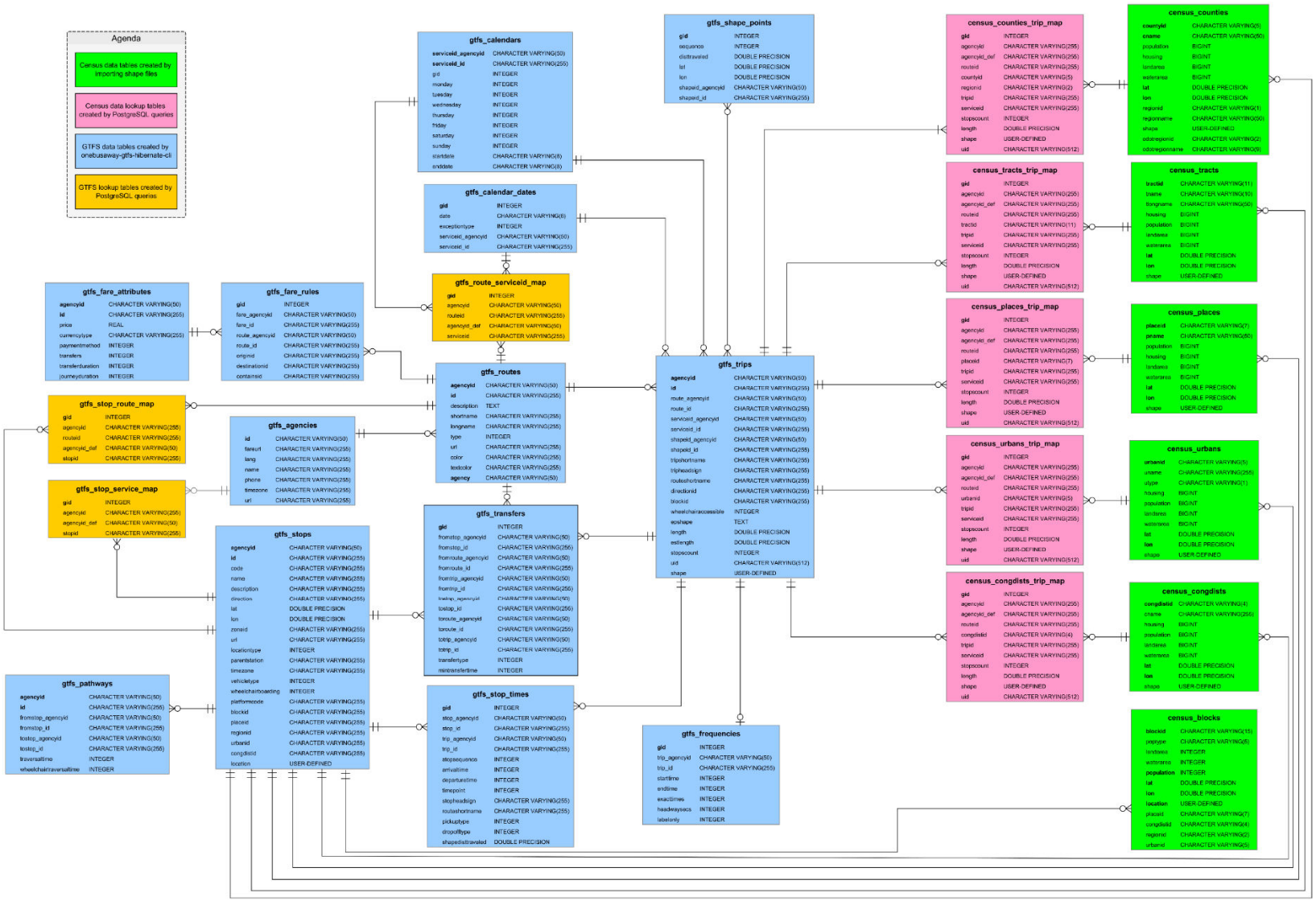


Figure 3.1: Entity-Relationship (ER) diagram of the PostgreSQL database

3.3.1 GTFS Data Tables

The GTFS data tables have a one-to-one correspondence to the fields of the files used by the GTFS specification. As mentioned earlier, the library `onebusaway-gtfs-hibernate-cli` creates these tables by reading the GTFS feeds and loading their data into the PostgreSQL relational database. The GTFS data tables can be identified in the ER diagram by the "gtfs_*" prefix in their names. More information on how GTFS data is organized into files can be found on Google's GTFS standard reference page *(25)*. The main GTFS data tables are:

- **gtfs_agencies**. Contains transit agency names and other information such as phone number and time zone.
- **gtfs_stops**. Contains the list of stops served by all transit agencies in the PostgreSQL database.
- **gtfs_routes**. Contains the list of routes served by all transit agencies in the PostgreSQL database.
- **gtfs_stop_times**. Contains the list of stop times for all trips served by all transit agencies in the PostgreSQL database.
- **gtfs_trips**. Contains the list of all trips served by every route offered by all transit agencies in the PostgreSQL database.
- **gtfs_frequencies**. Provides frequency information for trips that do not have stop times in the gtfs_stop_times table. This is a more flexible scheduling alternative offered by the GTFS standard for trips that do not have exact stop times for each stop.
- **gtfs_transfers**. Provides transfer information between different route/trips served by every transit agency.
- **gtfs_fare_rules**. Contains a list of rules for computing fares for some trips served by some transit agencies.
- **gtfs_fare_attributes**. Contains a list of fare prices and some other information that can be used to compute the fare price for transit agencies.
- **gtfs_pathways**. Pathways is not part of the official GTFS standard. Pathways describe connectivity among different stops and stations *(26)*.
- **gtfs_shape_points**. Provides a list of trip shape points for all trips served by all transit agencies.
- **gtfs_calendars**. Contains all the schedules used by transit agencies in the PostgreSQL database for scheduling their trips.
- **gtfs_calendar_dates**. Contains exceptions to service schedules in the gtfs_calendars table.

3.3.2 GTFS Data Lookup Tables

The GTFS data lookup tables are created manually by running queries in the PostgreSQL relational database to speed up more complex queries. The "gtfs_*" prefix and "*_map" suffix in a table name indicate a GTFS data lookup table in the ER diagram. The main GTFS data lookup tables are:

- **gtfs_stop_service_map**. Used to query stops for every transit agency or transit agencies that serve each stop.

- **gtfs_stop_route_map**. Used to query stops for every route or routes that serve each stop.
- **gtfs_route_serviceid_map**. Used to query service ids used for a route or routes that use a specific service id.

3.3.3 Census Data Tables

The census data tables are created by importing census data in shape file format into the PostgreSQL relational database. The prefix "census_*" in a table name identifies the census data tables in the ER diagram. The main census data tables are:

- **census_blocks**. Contains all census blocks with population greater than zero in the state of Oregon.
- **census_tracts**. Contains all census tracts within the state of Oregon.
- **census_counties**. Contains all counties within the state of Oregon, as well as ODOT transit regions.
- **census_places**. Contains all census places within the state of Oregon.
- **census_urbans**. Contains all urbanized area within the state of Oregon.
- **census_congdists**. Contains all congressional districts within the state of Oregon.

3.3.4 Census Data Lookup Tables

The census data lookup tables are created manually by running queries in the PostgreSQL relational database to speed up more complex queries. The "census_*" prefix and the "*_map" suffix in a table name indicate a census data lookup table in the ER diagram. The main census data lookup tables are:

- **census_tracts_trip_map**. Used to query all trip sections within a census tract or to query census tracts that contain a specific trip.
- **census_urbans_trip_map**. Used to query all trip sections within an urbanized area or to query urbanized areas that contain a specific trip.
- **census_counties_trip_map**. Used to query all trip sections within a county or to query counties that contain a specific trip.
- **census_places_trip_map**. Used to query all trip sections within a census place or to query census place that contain a specific trip.
- **census_congdistis_trip_map**. Used to query all trip sections within a congressional district or to query congressional districts that contain a specific trip.

4.0 TNA SOFTWARE TOOL DEVELOPMENT

This chapter describes the enhancements made to the TNA software tool once the PostgreSQL relational database was completed.

The rest of this chapter is organized as follows. Section 4.1 describes the development tasks completed on the *server* side of TNA software tool, whereas section 4.2 describes the development tasks completed on the *client* side.

4.1 SERVER SIDE MAIN DEVELOPMENT TASKS

One of the main advantages of a client-server architecture is that any resource intensive operations can be performed on the server side, thus enabling the best possible user experience on the client side. The TNA software tool takes maximum advantage of this benefit.

The majority of the development tasks presented in this section involved the design, implementation, and testing of data queries. The data retrieved by these queries can either be displayed directly onto the GUI of the TNA software tool or used to produce one of the many reports available through the TNA software tool. These data queries were designed to minimize data processing on the client side of the TNA software tool.

4.1.1 Loading GTFS Data into the Database

GTFS data can be loaded into the PostgreSQL relational database of TNA software tool by means of the open source library `onebusaway-gtfs-hibernate-cli`. The `onebusaway-gtfs-hibernate-cli` library was developed in Java and uses the library `Hibernate ORM` *(3)* to interface with various relational databases including MySQL, SQL Server, and PostgreSQL. More information about the `onebusaway-gtfs-hibernate-cli` library can be found on the OneBusAway developers' website *(7)*.

In order to import GTFS data into the PostgreSQL relational database of TNA software tool, the GTFs data need to first be copied to a local directory in the server. For example, the local directory `C:\Users\Administrator\Documents\Development\Feeds` was used as the default path in this project. Then, by issuing the command shown below in a web browser, the GTFS data are loaded into the PostgreSQL relational database:

```
http://server:port#/TNAtoolAPI-Webapp/modifiers/dbupdate/addfeed?feedname=gtfs.zip
```

Note that the field `server:port#` needs to be replaced with the server name/IP address and the port number the server is running on (e.g., `localhost:8080`). Also, the field `gtfs.zip` needs to be replaced with the specific name of the GTFS file name being loaded (e.g., `salem-or-us.zip`).

4.1.2 Web Application for the TNA Software Tool

A new web application (WebApp) was developed for the TNA software tool. The new WebApp is based on Java servlet, Jackson, and Jersey. Java servlet is a Java program that allows the

application to process requests received from the client application and generate responses. Jackson is a high performance open source JavaScript Object Notation (JSON) processor which is used on the webApp to create JSON objects. Jersey is an open source framework for developing RESTful web services in Java.

The new WebApp includes modules for adding/updating GTFS data in the PostgreSQL relational database (`com.webapp.api.modifiers`), generating JSON objects in response to requests from the GUI (`com.webapp.api.model`), and methods for generating responses to the queries from the GUI (`com.webapp.api`). The source code for the methods of the WebApp can be found in the module "TNAtoolAPI-Webapp".

4.1.3 Querying and Processing Spatial Data

The new WebApp of the TNA software tool must query spatial data (e.g., a list of census tracts) and issue spatial queries (e.g., identify census block internal points within a user defined shape). This functionality is not supported by the library `onebusaway-gtfs-hibernate`. Therefore, a new library was developed to enable these required actions.

The library `library-hibernate-spatial` uses `Hibernate Spatial` and `Hibernate` to connect to the PostgreSQL relational database and relies on `GeoTools` to manipulate geographic data including the creation of geographies such as point and polyline as well as performing projections. The library `library-hibernate-spatial` consists of several packages. The `com.library.model` package uses Plain Old Java Objects (POJO) to map to corresponding data tables in the PostgreSQL relational database. The `com.library.util` package has a class for managing connections to the PostgreSQL relational database, and the `com.library.samples` package has an `EventManager` class that queries spatial data from the PostgreSQL relational database.

As explained in the documentation for `Hibernate Spatial` and `Hibernate` *(3, 4)*, an XML mapping file is used to map POJOs to the data tables in the PostgreSQL relational database. The XML mapping file is available via the following file.

```
library-hibernate-spatial/src/main/resources/mapping.hbm.xml
```

The line shown above indicates the location of the file within the source code of the project. Given this address, a developer can find the file regardless of how their system is configured.

All queries used to retrieve spatial data from the PostgreSQL relational database are also stored in the same file.

4.1.4 Onebusaway-gtfs-hibernate Library

The new WebApp of the TNA software tool uses the library `onebusaway-gtfs-hibernate` to query GTFS data from the PostgreSQL relational database. The library `onebusaway-gtfs-hibernate` includes a few standard queries that are used by the OneBusAway project for trip planning.

However, the TNA software tools requires many more queries to retrieve very detailed data to generate reports and on map visualizations. Therefore, the POJOs in the library `onebusaway-gtfs` had to be modified to accommodate new attributes. Also, the `Hibernate` object mapping file was updated to match the changes made in the database schema of the PostgreSQL relational database and the POJOs. The XML mapping file is stored in:

```
onebusaway-gtfs-hibernate/src/main/resources/org/onebusaway/gtfs/model/GtfsMapping.hibernate.xml
```

The queries are stored in a separate XML file:

```
onebusaway-gtfs-hibernate/src/main/resources/org/onebusaway/gtfs/impl/HibernateGtfsRelationalDaoImpl.hibernate.xml
```

4.1.5 Multi-Database Feature

One of the desired features of the TNA software tool is the ability to access and compare GTFS data and census data from different time periods (e.g., first quarter versus last quarter of a given year). To enable this functionality, several instances of the PostgreSQL relational database can now be stored in the server and accessed by the TNA software tool. The different instances of the PostgreSQL relational database must be created manually on a specific schedule (e.g., quarterly or yearly).

An instance of the PostgreSQL relational database available through the TNA software tool may contain customized GTFS and shape data. However, each individual instance of the PostgreSQL relational database must store a complete set of census data available through the US Census Bureau (e.g., 2010 census data is currently used by the TNA software tool). This is important to note because there are data tables in the PostgreSQL relational database that are created based on both transit (i.e., GTFS and shape data) and census data and have relationships with both transit and census data tables.

The TNA software tool has two libraries that access the PostgreSQL relational database (i.e., `onebusaway-gtfs-hibernate` and `library-hibernate-spatial`). The library `onebusaway-gtfs-hibernate` accesses GTFS data using `Hibernate`, which uses an XML configuration file that contains the URL, username, password, and type of each instance of the PostgreSQL relational database. An individual XML configuration file is needed for each individual instance of the PostgreSQL relational database. The `Hibernate` configuration files for the library `onebusaway-gtfs-hibernate` are stored in:

```
onebusaway-gtfs-hibernate/src/test/resources/org/onebusaway/gtfs/examples/
```

The XML configuration files for the library `library-hibernate-spatial` are stored in:

```
library-hibernate-spatial/src/main/resources
```

The WebApp uses the information stored in a Java object to find the location and the quantity of the `Hibernate` configuration files, as well as the name of each instance of the PostgreSQL

relational database that will be displayed on the GUI. Database names are currently displayed as "Database1" and "Database2", but they can be changed to more meaningful names. The Java object that contains all the information described above is located at:

```
onebusaway-gtfs/src/main/java/org/onebusaway/gtfs/impl/Databases.java
```

4.1.6 Methods for On-Map Reporting

The TNA software tool generates a variety of visual reports directly on the GUI referred to as *on-map* reports. On-map reports can be generated for a specific geographical area defined by a user-drawn geometric shape (e.g., circle, rectangle, or polygon), or by clicking on single stops cluster icons (i.e., stops clusters that represent a single stop) on the main map area. On-map reports are based on transit data (e.g., stops, routes, schedules, etc.) and on geospatial data (e.g., census tracts, census block internal points, etc.)

A method was developed in the WebApp to invoke other methods in the libraries `onebusaway-gtfs-hibernate` and `library-hibernate-spatial` modules to acquire the GTFS and census data required for generating on-map reports. The results are then presented as JSON objects and sent to the GUI for displaying.

4.1.7 Methods for New Reports

Several new reports were added to the TNA software tool. Summary and extended reports for different geographic areas (e.g., counties, census places, congressional districts, etc.) are available in the TNA software tool. Each report is generated by an individual method in the WebApp on the server side of the TNA software tool.

4.1.8 Changes Made to Existing Features of the TNA Software Tool

Several features of the TNA software implemented with methods from the OpenTripPlanner (OTP) framework had to be re-programmed to make them compatible with the new software architecture and, in particular, with the PostgreSQL relational database.

The following sections describe the new methods that were developed. These methods were implemented along with queries in the PostgreSQL relational database and stored inside the WebApp module.

4.1.8.1 Hierarchical List of Transit Agencies

This method is used to provide the data required to populate the Oregon Transit Agencies (OTAs) floating dialog box in the GUI of the TNA software tool. The OTAs floating dialog box uses a tree-like menu that lists transit agencies at its top level, followed by routes and then trips, as depicted in Figure 4.1.

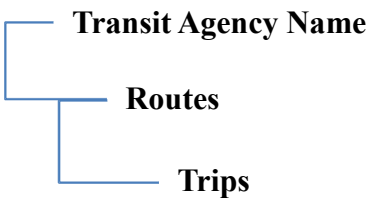


Figure 4.1: TNA software tool transit agency pane tree structure

Since there are usually many trips with an identical shape (i.e., sequence of stops) but different schedules, only trips with unique shapes are added to the tree-like menu to keep the size of the OTAs floating dialog box manageable. Also, since "trip_short_name" is an optional field in the *trips.txt* file of the GTFS specification many agencies do not use it. In such cases, the field "trip_headsign" (i.e., another optional filed in trips.txt file) is used. If neither "trip_short_name" nor "trip_headsign" exist, the pattern "From+ first stop name + "to"+ last stop name" is used to generate a name for the trip (e.g. From SE Jackson & Main to NE 11th & Holladay).

Querying all transit agencies along with their route and trips, as well as identifying and naming trips with unique shapes, are very resource intensive and time consuming tasks that can take minutes to complete. To improve processing speed and to reduce the load on the server, all instances of the transit agency menu for all PostgreSQL relational databases are generated once and then stored in memory.

Since the library `jstree` is used to populate the OTAs floating dialog box on the GUI of the TNA software tool, a JSON object formatted according to the `jstree` input specifications is an ideal form for providing data to the tree-like menu. JSON is a lightweight data interchange format which is easy to read and write for both humans and computers *(27)*. More details on how the tree-like menu was implemented is available in section 4.2.1.

4.1.8.2 Stops by Agency ID

This method provides the name and coordinates of the stops for a given transit agency identified by its agency ID.

It is important to note that there are a few Oregon GTFS feeds that contain data for multiple transit agencies. In these cases, all the stops contained in the multi-agency GTFS feed will have a default agency ID which is assigned by the library `onebusaway-gtfs-hibernate-cli`. Therefore, filtering stops by their agency IDs may not always work since the stops identified by the agency IDs other than the default agency ID cannot be retrieved.

To query the stops by agency ID in the prior version of the TNA software tool, all the routes for the given transit agency were queried first and then the stops for every route were queried and copied to a list. Since a single stop can be served by multiple routes, every stop was added to the list only if it was not already in it. This resource intensive querying procedure was the only way to implement this functionality in the OTP framework. In the version TNA software tool based on the PostgreSQL relational database, a lookup table is created (i.e., `gtfs_stop_service_map`).

The lookup table contains agency ID, default agency ID, and stop ID. Therefore, all stops for a given agency can be quickly queried by their *real* agency IDs.

4.1.8.3 Stops by Route

This method provides a list of stops for a given route and transit agency and it is used when a route is clicked on the tree structure of the OTAs floating dialog box to retrieve stop coordinates and stop names and to display them on the main map interface.

To make the method more efficient, a lookup table (i.e., `gtfs_stop_route_map`) is created which contains agency ID, default agency ID, route ID, and stop ID. The lookup table allows for fast and efficient querying of stops by routes.

4.1.8.4 Route Shape by Trip

This method provides the shape for a trip given agency ID, route ID, and trip ID. This method is used when a trip is selected on the OTAs floating dialog box to be displayed on the main map interface and returns the coordinates of the route shape in the form of an *encoded polyline*. The specifications and algorithm of an *encoded polyline* are explained in the Google Maps API *(28)*. The encoded polyline is then converted into L.LatLng objects using the library `Leaflet.encoded` so that it can be displayed on a map.

In order to make the data selection query more efficient, the shapes for every trip are queried from the data table `gtfs_shape_points` and encoded using an implementation of Google encoded polyline algorithm in PostgreSQL. For trips without shape data, a set of stop coordinates sorted by the stop sequence is used instead of shape points (since shape data is optional in the GTFS standard). The encoded shape data is stored in a new column in the data table `gtfs_trips` of the PostgreSQL relational database.

4.1.8.5 Transit Agency Summary Report

This method generates the "Transit Agency Summary" report in JSON format. This method has no input parameters, and once it is called, it generates a list that includes the agency ID, agency name, phone number, count of total routes, count of total stops, and fare data, for all the transit agencies in Oregon for which a GTFS feed is available.

In the new version of the TNA software tool, the average fair and the median fair are added to the "Transit Agency Summary" report in addition to the fare URL.

4.1.8.6 Transit Agency Extended Report

This method generates the "Transit Agency Extended" report in JSON format. This method generates a list that includes, for a given transit agency, the fields agency ID, agency name, route miles, route stops, stops per route mile, population served by route, service miles, service stops, population served by service, and service hours. This report can be generated for a single date or a selection of multiple dates.

4.1.8.7 Routes Report

This method generates the "Routes" report. This method generates a list that includes, for a single route of a given transit agency, route ID, route name, route long name, route type, route length, total stops count for every route, unduplicated population, service stops, population served by service, and route description. This report can be generated for a single date or a selection of multiple dates.

4.1.8.8 Stops Report

This method generates the "Stops" report. This method generates a list that includes, for a single stop of a single route of a given transit agency, stop ID, stop name, the route(s) that the stop belongs to, and population served.

Agencies like TriMet (with more than 8,000 stops) make the process of report generation very resource intensive and time consuming. To improve the user experience and to decrease unnecessary load on the server, the fields *population served* and *routes* associated with a stop are loaded after the report is generated. Also, population and route data are computed only for the portion of the report that fits into a single page of the report (if the full report takes more than one page). This allows the report to load faster, since it does not have to wait until all population numbers and routes are computed and additional population/routes are calculated only if necessary.

4.1.8.9 Population within X Distance of Stops

This method computes the population around a point (given its coordinates) within a certain distance *X* (in miles) from either a stop, a route, or a transit agency. In the new version of the TNA software tool, census block internal points along with the population living in each census block are loaded into the PostgreSQL relational database. A spatial query is then used to find the centroids of the census block internal point within a given distance of a given point. This method is implemented in the library `library-hibernate-spatial` and many methods in the WebApp use it for population computations.

4.1.8.10 Route Miles

Route miles is calculated as the summation of the lengths of all routes operated by a given agency. Since the length of a route can vary based on the service schedule, the length of the longest trip is considered in this computation. In order to speed up the process of querying data from the database, the length of each trip is computed based on the length of the trip shape stored in the trips table and added to the trips table.

4.1.8.11 Service Miles per Day

In order to compute the service miles for a given date, the parameters *number of trips* and *length of every trip* are queried for the given day of the week and then summed up over all routes.

The diagram depicted in Figure 4.2 illustrates how the GTFS specification relates route and trip data to daily schedules and calendar information. Every route for a given transit agency is identified by a unique route ID. For a given route ID, there has to be at least one trip ID that matches the information related to arrival time, departure time, and stop sequence in the file *stop_times.txt*. Therefore, trips identified by unique trip IDs, are different instances of the same route being served on different hours in the file *stop_times.txt*. The attribute "service_id" in the file *calendar.txt* is the key to match trip IDs with different days of the week. For every service pattern represented by a set of days (e.g., Monday, Wednesday, Thursday, and Friday) that some service is active on, there needs to be a unique "service_id" that can be matched to one or more trip IDs.

In addition to the file *calendar.txt*, which specifies regular transit agency schedule, there is another file named *calendar_dates.txt* to specify exceptions in the regular schedule. However, the GTFS standard allows transit agencies to define all their regular schedule as exceptions and not by using the file *calendar.txt*. Therefore, in the new version of the TNA software tool, both of these files (i.e., *calendar.txt* and *calendar_dates.txt*) are used to assess the availability of trips for a given date.

To compute service miles per day for a given transit agency, all the routes along with their corresponding trip IDs and service IDs are queried. Once this information is obtained, service IDs are compared to calendar data for the given day to find which service IDs are in effect for the day of interest. Finally, the trip lengths for the effective trips on the given day are computed and summed up to find service miles per day.

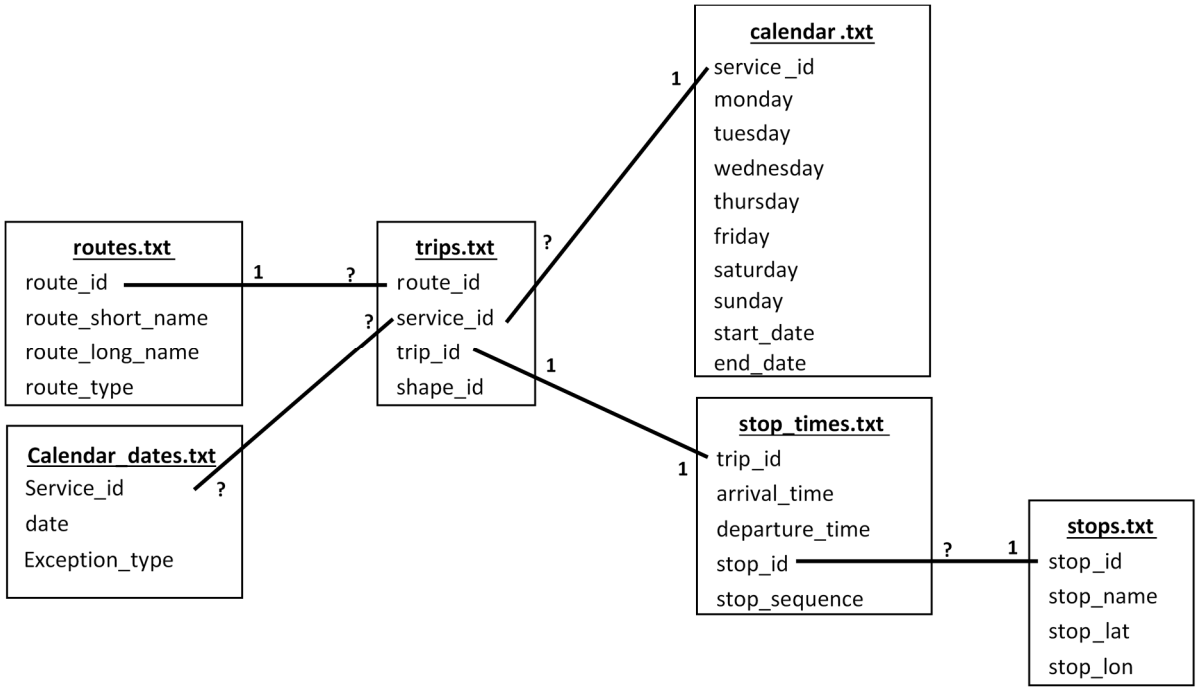


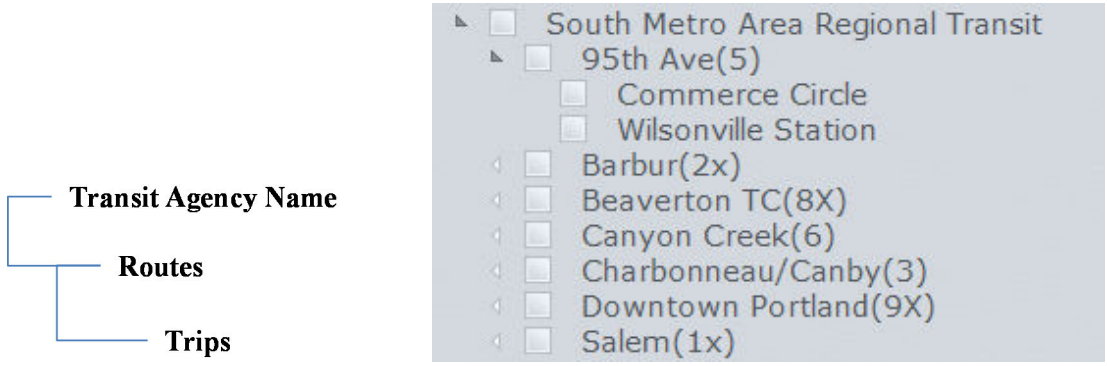
Figure 4.2: Relationships among routes, trips, calendar data and schedules in the GTFS data

4.2 CLIENT-SIDE DEVELOPMENT TASKS

The GUI of the previous version of the TNA software tool was used as the basis for developing the GUI of the new TNA software tool. The main idea was to build a new application programming interface (API) which is compatible with the older version of the TNA software tool and add new features. Therefore the new GUI uses the same RESTful API and all the former API commands are available in the new version.

4.2.1 The Oregon Transit Agencies Floating Dialog Box

Figure 4.3 depicts the Oregon Transit Agencies (OTAs) floating dialog box that is used in the TNA software tool to store the transit agency tree-like menu. The OTAs floating dialog box is implemented using the libraries `jQuery UI` and `jQuery UI dialogextend`. The OTAs floating dialog box is transparent to allow tracking of changes that take place on the main map interface when the contents of the OTA floating dialog box are clicked. The OTAs floating dialog box can also be moved, resized, minimized, collapsed, and maximized.



(a) Pane Tree Structure (b) South Metro Area Regional Transit Agency Example

Figure 4.3 The hierarchical structure of the extended OTAs floating dialog box

Inside the OTAs floating dialog box, a tree-like menu is implemented to view and select one or multiple transit agencies for visualization. The transit agency tree-like menu has three levels:

- **Level 1**. Lists all transit agencies in the state of Oregon in alphabetical order. Clicking an item at this level unchecks all of its child nodes (if checked) and displays the stops that belong to the selected transit agency on the map. Right clicking on a transit agency name opens a menu that allows displaying/hiding all trip shapes served by the transit agency.

- **Level 2.** Lists all the routes that belong to the selected transit agency (parent node). Selecting any nodes at this level unchecks its parent node (if checked) and displays the stops that are served by the selected route. Right clicking on a route name opens a menu that allows displaying/hiding all trip shapes served by the route.

- **Level 3.** Lists all trips with unique shapes for the route (expanded parent node). Checking a node at this level, displays the trip shape on map.

The open source JavaScript plug-in `jstree` was used to implement the tree-like menu used in the OTAs floating dialog box. The `jstree` plug-in is customized so that any of the nodes in the tree-like menu can be selected by selecting one of the available checkboxes. Selecting a checkbox in the tree-like menu triggers an event based on the type of node selected (i.e., either a transit agency, a route, or a trip). In order to populate the tree-like menu, a JSON object is created on the server side application (see section 4.1.1). Therefore, no extra manipulation is required for data conversion, which ensures a swift experience on the client side of the TNA software tool. The tree-like menu is loaded once the client side main page is called and it is used as long as the page open remains open.

Right clicking on levels 1 and 2 objects on the tree-like menu allows the display of all trip shapes on the main map interface. However, only the shape of the longest trip is displayed on the main map interface.

4.2.2 Location Search Box

The location search box allows for searching specific locations based on their names (e.g., cities, street addresses, places and businesses, etc.) The location search box is implemented using the OpenStreetMap (OSM) API which is free and open source.

4.2.3 Map Tiles

The OSM layer and the Toner layer by stamen design *(29)* were available in the earlier version of the TNA software tool. The toner map tiles provides improved visibility of a transit agency characteristics (i.e., stops and routes) for visualization and printing purposes.

Additionally, the Google Aerial photography map tiles replace the OSM Aerial map tiles used in the former version of the TNA software tool to provide higher quality imagery using Google API.

4.2.4 Map-in-map Feature

The map-in-map feature provides a broader view of the area that is being displayed in the main map interface. The map-in-map feature is implemented with the open source JavaScript library `Leaflet MiniMap` and utilizes an orange rectangle to indicate which region of the state of Oregon (or other large geographical area) is being displayed in the main map interface. The map-in-map feature can also be used for moving over large areas and can be minimized by clicking on the arrow icon on its lower right corner.

4.2.5 Displaying Stops and Route Shapes

As described in section 4.1.8.2 and section 4.1.8.4, methods were developed on the server to provide stops and trip shapes to the client side of the TNA software tool. The JavaScript library `Leaflet` is used to implement all the visualization capabilities of the TNA software tool.

When displaying stops, the list of stop names and coordinates are provided. The trip shapes are received in encoded polyline form from the server and, therefore, need to be decoded into coordinates in order to be displayed on the main map interface using Leaflet. `Leaflet.encoded` is a plug-in for leaflet that adds the capability to encode and decode polylines to `Leaflet`.

4.2.6 Clustering Stops

The TNA software tool displays stops as clusters that change in size as the level of zooming is changed. The software component used to implement this functionality was the plug-in library `Leaflet markercluster`.

When the mouse is hovered over stops clusters that represent more than two stops, a convex polygon (convex hull) shows the area covered by the transit stops present in the cluster. In the new version of the TNA software tool, the stops cluster becomes transparent when the polygon is displayed to provide more contrast and better visibility.

4.2.7 Reports

The new version of the TNA software tool has an improved reports interface with many new reports added. Reports are now displayed in a tabular format with various new features. The library `jQuery UI DataTables` is used to display report content. This library allows reports to be sorted, searched, and presented in several pages to improve usability. The library `jQuery UI DataTables` also supports copying reports to the clipboard as well as exporting them in PDF and CSV formats.

With the exception of the “Transit Agency Summary” report and the “Stops” report, all the transit agency reports in the TNA software tool include a full calendar and a date picker tool. As discussed in section 2.2.8.11, reporting based on the day of the week (which was used in the earlier version of the TNA software tool) did not work for all transit agencies because of different implementations of the files *calendar.txt* and *calendar_dates.txt* in the GTFS standard. With the full calendar and date picker, a report can be generated for a specific date or a selection of dates. Also, the generated reports are more accurate since the exceptions in service are also taken into account.

The **Reports** button displayed on the upper right corner of the OTAs floating dialog box opens a menu that shows a list of all available reports in the TNA software tool.

The option *Transit Agency Reports* provides access to the “Transit Agency Summary” report, which provides general information on all transit agencies in the state of Oregon, as well as access to the "Routes" report, “Schedule” report, "Stops" report, and "Transit Agency Extended"

report. As mentioned in section 2.2.8.5, average fare and median fare information were added to the "Transit Agency Summary" report.

Other report options include the *Counties Reports*, *Census Places Reports*, *Congressional Districts Reports*, *Urban Areas Reports*, and *ODOT Transit Regions*. Clicking on any of the mentioned report categories opens summary level reports which presents a list of the selected geographic areas in the state of Oregon. Clicking on the Geo ID of the geographic areas in the list will open the extended reports for the selected area to provide more in depth metrics.

4.2.8 Displaying Geographical Shapes on the Main Map Interface

The TNA software tool can display the shapes of all Oregon counties, ODOT transit regions, urbanized areas (i.e., urban areas with population over 50,000), and congressional districts as a map layer. The corresponding shapes were obtained from the US Census Bureau *(24)* in *.shp* format (i.e., the native format provided by the U.S. census bureau) and imported into the PostgreSQL relational database. The shape data were then queried and converted to GeoJSON format (i.e., the format that the library `Leaflet` supports) using a PostgreSQL query. The resulting file was fed to the online mapshaper tool *(30)* to reduce the size of the layer and make it more suitable for visualization purposes by removing a portion of the points from the shapes and also discarding a few decimal points off the coordinate values. This process results in less accurate but very small shape files that can be easily overlaid on top of many layers such as stops and trip shapes on the main map interface of the TNA software tool. Every geographic area shape includes name and surface area (in square miles) which are displayed in an `L.control Leaflet` object that shows up when the mouse is hovered over a county shape on the map.

All shape layers are styled to be transparent to allow other layers beneath it to remain visible. There are also functions that slightly highlight the shapes and update the data displayed in the box over the map when the mouse is hovered over a shape. If a shape is clicked on with the mouse, the map zooms in on the corresponding area.

4.2.9 Linking Reports

There are 17 reports available in the TNA software tool. The "Transit Agency Summary" report is the main report in the *Transit Agency Reports* category which provides access to the other reports by clicking on either the agency ID, routes or stop hyperlinked numbers. The “Counties Summary” report provides access to census tracts and counties extended reports by clicking on the field Tracts Count and the field Geo ID, respectively. All other geographic area summary reports provide access to their corresponding extended reports by clicking on the field Geo ID.

To implement this feature, a hyperlink tag was added to the corresponding fields in the parent report (i.e., the report that provides access to a lower level report). Since these links are not actual hyperlinks (i.e., they do not refer to a URL), a JavaScript function was implemented to capture the click events. Two custom attributes (i.e., Type and ID) were necessary to identify the type and parameters of the report that needed to be loaded. Therefore, each time a hyperlink is clicked, the JavaScript function decides which type of report to load by looking into the type

attribute and then passes the appropriate parameters to the query that generates the report by looking into the ID attribute.

4.2.10 On-Map Report

The on-map report can be invoked by either drawing a geometric shape (circle, rectangle, or a polygon) on the main map interface or by clicking on a single stop cluster to reveal a window with a button to generate the on-map report.

The library `LefLet.draw` is used for drawing, editing, and deleting shapes on the main map interface. Once the shape is drawn, it is sent to the server to generate the on-map report. The contents of the on-map report are displayed using a `jQuery UI` dialog box on the main map interface. Invoking an on-map report hides the other layers (e.g., stops, routes, etc.) and collapses the OTAs floating dialog box.

4.2.11 Multi Database Functionality

The multi database feature is implemented on both the visualization and reporting interfaces of the GUI of the TNA software tool. A new button with a gear icon was added to the top right corner of the OTAs floating dialog box to specify the database. When the TNA software tool loads, the first database in the list is chosen as the default database. Changing the database on the visualization interface reloads the page (which removes all visualized objects such as stops, routes, and on-map reports from the map), re-populates a new instance of the OTAs floating dialog box based on the new database, and all new visualizations and on-map reports will be from the new database from this point. The multi database feature is also available in the reports interface through a menu box on the top right corner of every report. Changing the database on any report, reloads the report with the new values extracted from the selected database.

A method was developed in the WebApp to allow the GUI to retrieve available database names and populate the database list based on that information. To add or remove a database, the configuration files discussed in section 2.2.5 needs to be modified. Once the configuration files are modified, the GUI retrieves the new database information every time a new page is loaded (e.g., a report is opened or the main GUI is refreshed/opened).

To implement the multi database feature, a new parameter named “dbindex” was added to all API commands. This parameter is also visible in the address bar of the web browser at the very end of the URL. This is how the parameter is sent to another page while opening a new report.

5.0 CONCLUSIONS AND FUTURE WORK

The main objective of this project was to continue to enhance the capabilities of the prototype TNA software tool created in the first phase of the project. Accomplishing this objective required a major change in the underlying architecture of the TNA software tool. The most important change in the architecture involved replacing the "graph file" used by OpenTripPlanner (OTP) with a PostgreSQL relational database. The PostgreSQL relational database added a significant level of flexibility when loading, organizing, and querying not only GTFS data feeds but also population data.

The enhance TNA software tool incorporates the GTFS feeds of 49 different Oregon transit agencies and can produce reports based on transit agency, county, census place, congressional district, urban area, or ODOT transit region. Each of these reports categories also offer additional reporting capabilities to the user. Several reports are also available through its graphical user interface. The TNA software tool will also provide data that can be imported into the Oregon Communities Reporter *(31)*.

The necessary documentation and source code to install and run an instance of the TNA software tool is currently available at GitHub *(32)*. This documentation and source code will be updated as the project progresses and additional enhancements to the TNA software tool are completed.

6.0 REFERENCES

1. Porter, J., D. Kim, and S. Ghanbartehrani. *Proof Of Concept: GTFS Data As A Basis For Optimization Of Oregon'sRegional And Statewide Transit Networks*. Publication SPR 752. May 2014.
2. The Eclipse Foundation. Eclipse - The Eclipse Foundation open source community website. https://www.eclipse.org/. Accessed Jan. 30, 2014.
3. Red Hat Projects Community. Hibernate ORM. http://hibernate.org/orm/. Accessed Dec. 7, 2014.
4. Geovise open source GIS solutions. Overview | Hibernate Spatial. http://www.hibernatespatial.org/. Accessed Dec. 23, 2014.
5. Onebusaway project. onebusaway-gtfs - About. http://developer.onebusaway.org/modules/onebusaway-gtfs-modules/1.3.4-SNAPSHOT/onebusaway-gtfs/index.html. Accessed Dec. 23, 2014.
6. Onebusaway project. onebusaway-gtfs-hibernate - About. http://developer.onebusaway.org/modules/onebusaway-gtfs-modules/1.3.4-SNAPSHOT/onebusaway-gtfs-hibernate/index.html. Accessed Dec. 23, 2014.
7. Onebusaway project. onebusaway-gtfs-hibernate-cli - About. http://developer.onebusaway.org/modules/onebusaway-gtfs-modules/1.3.4-SNAPSHOT/onebusaway-gtfs-hibernate-cli/index.html. Accessed Dec. 23, 2014.
8. Open Source Geospatial Foundation. GeoTools Documentation — GeoTools Documentation. http://docs.geotools.org/. Accessed Dec. 23, 2014.
9. World Wide Web Consortium. W3C HTML. http://www.w3.org/html/. Accessed Jan. 30, 2014.
10. World Wide Web Consortium. Cascading Style Sheets. http://www.w3.org/Style/CSS/Overview.en.html. Accessed Jan. 30, 2014.
11. Flanagan, D. *JavaScript: the definitive guide*. O'Reilly Media, Inc., Beijing, 2011.
12. The jQuery Foundation. jQuery. http://jquery.com/. Accessed Sep. 1, 2013.
13. The jQuery Foundation. jQuery UI. http://jqueryui.com/. Accessed Sep. 3, 2013.
14. SpryMedia Ltd. DataTables: Table plug-in for jQuery. http://www.datatables.net/. Accessed Dec. 23, 2014.
15. ROMB. Jquery-dialogextend. http://romb.github.io/jquery-dialogextend/. Accessed Dec. 23, 2014.
16. Otto (@mdo), M., and Jacob (@fat). Bootstrap. http://getbootstrap.com/javascript/#dropdowns. Accessed Dec. 23, 2014.
17. Bozhanov, I. jsTree » Home. http://www.jstree.com/. Accessed Aug. 26, 2013.
18. Agafonkin, V. Leaflet - a JavaScript library for mobile-friendly maps. http://leafletjs.com/. Accessed Sep. 3, 2013.
19. Timberlake, A. Leaflet/Leaflet.markercluster · GitHub. https://github.com/Leaflet/Leaflet.markercluster. Accessed Sep. 3, 2013.
20. Jan Pieter Waagmeester. Leaflet.encoded. https://github.com/jieter/Leaflet.encoded. Accessed Aug. 25, 2013.
21. Nordan, R. Norkart/Leaflet-MiniMap · GitHub. https://github.com/Norkart/Leaflet-MiniMap. Accessed Dec. 23, 2014.

22. Google Inc. Google Maps JavaScript API v3 - Google Developers. https://developers.google.com/maps/documentation/javascript/tutorial. Accessed Dec. 23, 2014.
23. OpenStreetMap Project. API - OpenStreetMap Wiki. http://wiki.openstreetmap.org/wiki/API. Accessed Dec. 24, 2014.
24. United States Census Bureau. TIGER Products - Geography - U.S. Census Bureau. http://www.census.gov/geo/maps-data/data/tiger.html. Accessed Sep. 3, 2013.
25. Google Inc. General Transit Feed Specification Reference - Transit — Google Developers. https://developers.google.com/transit/gtfs/reference#frequencies_fields. Accessed Sep. 4, 2013.
26. Hughes, J. Proposal: Provide a way to indicate entrances and pedestrian pathways - Google Groups. https://groups.google.com/forum/#!msg/gtfs-changes/Gk88QYQYgtw/UNdVCH0v-RIJ. Accessed Dec. 24, 2014.
27. ECMA International. Introducing JSON. http://www.json.org/. Accessed Sep. 14, 2013.
28. Google Inc. Encoded Polyline Algorithm Format - Google Maps API — Google Developers. https://developers.google.com/maps/documentation/utilities/polylinealgorithm. Accessed Aug. 25, 2013.
29. Stamen Design. stamen design | Dotspotting's Toner Cartography available for download. http://content.stamen.com/dotspotting_toner_cartography_available_for_download. Accessed Sep. 3, 2013.
30. Bloch, M. mapshaper. http://mapshaper.org/. Accessed Dec. 24, 2014.
31. Oregon State University. Communities Reporter. http://oe.oregonexplorer.info/rural/CommunitiesReporter/. Accessed Jan. 6, 2015.
32. Ghanbartehrani, S., and A. Mohseni. tnatool/test · GitHub. https://github.com/tnatool/test. Accessed Jan. 6, 2015.

APPENDIX A: GLOSSARY OF TERMS

Service stops

The number of trips scheduled at a stop in a route[1]. The service stops for a route is calculated as its stop count multiplied by the number of visits per stop.

Example: A single route with 12 stops and two visits per stop would have a service stops of 12 * 2 = 24.

Population served by service (see service stops)

Total unduplicated population impacted within an *X*-mile radius (i.e., stop distance) of all stops on a trip or trip segment specified.

Route population served by service is calculated as route service stops multiplied by the unduplicated population within an *X*-mile radius (i.e., stop distance) of all stops on a trip or trip segment specified.

Transit agency population served by service is the sum of population served by service over all routes.

Example: A route with 12 stops, two visits per stop, and an unduplicated population of 3,400 within 0.25 miles of all stops would have a population served by service of
12 * 2 * 3400 = 81,600.

Region

A geographical boundary that defines a state, county, census tract, census block, census place, state congressional district, or federal congressional district used for calculating performance metrics.

Round Trip

A two-way pass of all stops in a route, accounting for *both* outbound and inbound travel.

Route

A collection of ordered stops presented publicly as a "line" with a name [1].

Service Miles Per Day

Total miles driven by a transit agency in a day over all round trips of a route. Service miles may be calculated based on a seven day week as well.

Example: An agency has three routes that measure 10 miles, 12 miles, and 25 miles in length. Each route has one round trip per day. The total service miles per day for this agency is (10 + 12 + 25) * 2 = 94 miles.

[1] Not a measure of actual times a bus stops, but a measure of possible stops.

Stop

Designated location on a route where transit vehicles pick up customers.

Trip

A one-way pass of all stops in a route, accounting for *either* outbound or inbound travel.

Trips Per Day

Number of trips in a single day for a given route.

Unduplicated Stop Population

Unduplicated population count within an *X*-mile radius (i.e., stop distance) of a stop. Population information is taken from 2010 census block data from the U.S. Census Bureau *(24)*.

To calculate the unduplicated stop population, the population of a census block will be assumed to be concentrated at the geographic centroid of the census block shape. Then, the population within *X* miles of a stop will be calculated as the sum of the population of the census blocks whose centroids are contained within the area defined by the *X*-mile radius from the stop.

Visits per stop

Number of trips per stop on a route.

Average Fair

Average of reported fair prices in fair_attributes.txt file. Since fair_attributes.txt file is optional, "NA" is reported in case no fair data is available.

To calculate average fair for a transit agency, fair prices for all routes served by the transit agency is queried and averaged. For a single route, fair price for that route is queried. For a geographic area, fair prices for all routes within the given geographic area is queried and averaged.

Median Fair

Median of reported fair prices in fair_attributes.txt file. Since fair_attributes.txt file is optional, "NA" is reported in case no fair data is available.

To calculate median fair for a transit agency, fair prices for all routes served by the transit agency is queried and sorted. The one in the middle (in case of an odd numbered list) or the average of the 2 fairs in the middle (in case of an even numbered list) of the fair list is reported. For a single route, fair price for that route is queried. For a geographic area, fair prices for all routes within the given geographic area is queried.

APPENDIX B: GEOGRAPHIC AREA REPORT METRICS DEFINITION

Metric	Definition	Required input parameters	Computation method	Sample Output
Stop count	Number of transit stops inside a given geographic area	Geographic area type	Filter transit stops based on their geocode. Filtering can be done by the type of geographic area such as census block(s), census tract(s), census place(s), ODOT transit region(s), congressional district(s), urban area(s), or the state of Oregon.	12 transit stops in census block No. 3071
Stops per square mile	Stop count in a given geographical area divided by the square miles of the geographical area	Geographic area type	The area (in square miles) of census blocks, census tracts, census places, ODOT transit regions, congressional districts, urban areas, and the state of Oregon can be derived from the shape files provided by the US census bureau. The square mileage of these areas can be derived by subtracting the (Land/water/total)	1.54 stops/mi^2
Population within *X*-mile radius of (individual) stops	Sum of population concentrated on census block internal points within a *X*-mile radius of a stop inside a given geographic area	Geographic area type, population search radius (in miles)	All census block internal points within an *X*-mile radius of the stop of interest in a given geographic area are identified and filtered by the geocode for the given geographic area.	4,534 people
% of population served	Unduplicated population within a *X*-mile radius of all the stops in the given geographic area divided by the population of the given geographic area	Geographic area type, population search radius (in miles)	All unique census block internal points within a *X*-mile radius of all stops in the given geographic area are identified and filtered by the geocode for the given geographic area. The population associated with these census block internal points is added and then divided by the population of the given geographic area.	25.3 %
Unserved population	100% minus % of population served	Geographic area type, population search radius (in miles)	100 - % of population served.	12.4 %

Metric	Definition	Required input parameters	Computation method	Sample Output
Average/Median fare	Average/Median of the fare over all the routes that operate within the given geographic area	Geographic area type	Stops located within geographic area of interest are identified. Then, all routes that serve stops within the given geographic area are queried. If fare information is available from the transit agencies that operate in the given geographic area, the average fare and median fare are calculated.	$4.25
Hours of service	The union of service time intervals for all stops within the given geographic area being served for the given date	Geographic area type, date	Earliest and latest stop times for all stops in the given geographic area are queried for the given date. Then, the union of service time intervals is computed.	8:00 AM – 6:00 PM
Service days	Set of days in which at least one stop in the given geographic area is being served within the seven days of the selected date	Geographic area type, date	Service availability of all stops in the given geographic area is checked within seven days of selected date and the union of dates with active service is returned.	5/21, 5/22, 5/23, 5/26, 5/27
Number of (directly) connected communities	List of the geographic areas of the same type that are connected to the area of interest through routes	Geographic area type, area of interest	Stops located within geographic area of interest are identified. Then, all routes that serve stops within the given geographic area are queried. Finally, all stops for each route are queried and the connected communities are found based on the geocodes of the stops.	Benton, Linn, Lane, Multnomah
Service miles	Service miles within the given geographic area for the given date	Geographic area type, date	The length of all trips within the geographic area of interest are found and then multiplied by the number of times that the trip is being served on the given date.	1,356.3 mi
Service miles per square mile	Service miles within the given geographic area for the given date divided by the total area (in square miles) of the given geographic area	Geographic area type, date	Service miles are computed as explained above and then divided by the total area (in square miles) of the given geographic area.	12 mi/mi^2

Metric	Definition	Required input parameters	Computation method	Sample Output
Service miles per capita	Service miles within the given geographic area for the given date divided by the total population of the given geographic area	Geographic area type, date	Service miles are computed as explained above and then divided by the total population of given geographic area.	1.34 mi/person
Stops per service mile	Number of stops within the given geographic area divided by the service miles within the given geographic area	Geographic area type, date	Number of transit stops within the given geographic area divided by the service miles within the given geographic area.	0.45 stops/mi^2
Route miles	Sum of the lengths of all trips within the given geographic area	Geographic area type, date	All trips are queried for the given geographic area and the lengths of the individual trip shapes (that are active on the given date) within the given geographic area are computed and summed up.	125 mi
Service stops	Total stops within the given geographic area multiplied by the number of times each stop is being served for the given date	Geographic area type, date	All trips within the given geographic area are queried. Then, the number of times the trip is available for the given date is computed and multiplied by the number of served stops that fall inside the geographic area.	175 stop visits
Population served by service	Sum of unduplicated population within *X*-miles of each transit stop multiplied by the number of times the stop is being served for the given date for the given geographic area.	Geographic area type, date, population search radius	The computation method is similar to that of stopportunity. However, instead of stops, unduplicated population within *X* mile of stops for every trip inside the given geographic area is considered.	4,365 (population access)
Number of networks (spatially	Given a connection definition, how many	Geographic area type, date,		6 networks when

Metric	Definition	Required input parameters	Computation method	Sample Output
connectivity)	networks are there? (1 network when all service is connected).	connection distance (two stops are "connected" when within connection distance of each other)		connection distance is .00001 miles (provide list of services in each network, display differentiated networks on map)
% of population served (at specified service level)	Unduplicated population within a *X*-mile radius of all the stops in the given geographic area (where stop is <visited count> or more times in a day) divided by the population of the given geographic area	Geographic area type, population search radius (in miles), visit count		15.3 %

APPENDIX C: PHASE II REQUIREMENTS

The table below presents the full list of desired features in the TNA software tool and their (projected) implementation phase. All requirements marked to be implemented in phase II are implemented in the latest version of the TNA software tool.

FEATURE	NOTES	Implementation Phase
OTP Analyst		
Integration with OTP Analyst	The most practical approach is to add a link to an instance of OTP Analyst in the TNA software tool GUI.	**II**
Route/Schedule Information as a Service		
Schedule table	It is not recommended to integrate a "Web Service" which serves third party websites with an analysis tool that provides service to known users (e.g. Analysts in ODOT PTD). An application that provides web service is prone to scalability and security issues that might interfere with the analysis capabilities.	**II**
Route map data	Need more clarification/example on this feature.	
COMMUNITY EXPLORER		
Stops per square mile	Assuming it is stops per square mile of area served.	**II**
Service miles per square mile	Assuming it is service miles per served area.	**II**
Miles of service per capita	Assuming it is miles of service per served capita.	**II**
% of population served	Assuming it is percent of some geographic area's (e.g. county, state, tract, etc.) population served	**II**
Stops per service mile	Service miles and stop counts are already available in reports.	**II**
Average fare	Fare data can be included in the TNA software tool in case a relational database is used as data repository.	**II**
Median fare	Can be derived from fare data.	**II**
Fare per mile	Assuming it is fare per service mile.	

FEATURE	NOTES	Implementation Phase
Service cost per mile	Based on transit agency service cost data.	
Service cost per hour	Based on transit agency service cost data.	
Maximum possible distance travelable within network	This feature can be implemented efficiently using a relational database.	
Maximum possible distance travelable using all networks	This feature can be implemented efficiently using a relational database.	
Rides per year (total rides and special needs) [per capita?]	Ridership data can be included in the TNA software tool in case a relational database is used as data repository.	
Number of communities connected to network (direct/indirect)	Shape data can be included in the TNA software tool in case a relational database is used as data repository.	II
Stopportunity (stops per route per day)	***Already implemented.***	***I***
REPORTING FUNCTIONALITY		
Comparison Over Time	Using a relational database makes it easier and more efficient to store different versions of data in the database.	II
Load experimental GTFS data by the end user	We currently have no interface on the client side application to allow end user loads (GTFS) data into the graph file/database.	
REPORTS FOR GEOGRAPHIC AREA(S)		
State	A geo-spatial database allows geographical operations required for this type of analysis easier and more efficient to implement.	**II**
Urban / rural		**II**
County		**II**
Census Place		**II**
Census Tract		**II**
State/Fed congressional districts		**II**
Stop count		**II**
Service miles		**II**
Route miles		**II**
Route stops		**II**

FEATURE	NOTES	Implementation Phase
Service stops		**II**
Service hours		**II**
Population within ***x*** distance of a route		**II**
Stopportunity		**II**
Popstopportunity		**II**
Service days		**II**
Hours of service		**II**
Un-served population		**II**
ODOT transit region		**II**
Population within ***x*** distance of a stop (both duplicate and unique) & within specified geographic area		
TRANSIT AGENCY(S)/SERVICES		
Service hours		**II**
Service days		**II**
Hours of service		**II**
Geographic area(s) served	A geo-spatial database allows geographical operations required for this type of analysis easier and more efficient to implement.	**II**
Schedule tables	Can have an instance of schedule tables on the TNA software tool reports for private use.	**II**
Based on service area from stop/route for geographic area y – un-served population		
Stops	***Already implemented.***	***I***
Service miles	***Already implemented.***	***I***
Route miles	***Already implemented.***	***I***
Population within x distance of a stop (both duplicate and	***Already implemented.***	***I***

FEATURE	NOTES	Implementation Phase
unique)		
Population within x distance of a route	***Already implemented.***	***I***
Stopportunities	***Already implemented.***	***I***
Popstopportunities	***Already implemented.***	***I***
Service Stops	***Already implemented.***	***I***
Route Stops	***Already implemented.***	***I***
Stops per route mile	***Already implemented.***	***I***
Service day selection	***Already implemented.***	***I***
MAP DISPLAY		
Population value within ***x*** distance of a stop	Already computed, needs to be displayed on map.	**II**
Route run frequency	Already computed, needs to be displayed on map.	**II**
Stopportunity	Already computed, needs to be displayed on map.	**II**
Routes with frequency less than/more than ***z***	Already computed, needs to be displayed on map.	**II**
Map display reports that lend themselves to graphic expression	It is not clear what reports are meant to be displayable on map.	
Allow end user manipulation of routes, stops, day of week, display style, content, etc. of display	Some parts of the features in this category is unclear (e.g. end user manipulation of routes, stops, and content). Manipulation of day of week, and display style can be done in phase II.	**II**
Stops	***Already implemented.***	***I***
Routes	***Already implemented.***	***I***
Routes/stops of specified agencies	***Already implemented.***	***I***
SPATIAL CONNECTIVITY		
Where ***transfers.txt*** exist, support limitation in assumed connection based on transfers.txt data		
Number of spatially connected networks given a connection	A geo-spatial database allows geographical operations required for this type of	

FEATURE	NOTES	Implementation Phase
definition of stops within distance ***w***	analysis easier and more efficient to implement.	
Allow a network connection definition of stops within same census place	A geo-spatial database allows geographical operations required for this type of analysis easier and more efficient to implement.	
MAP DISPLAY		
Route segments by service density		
Stop clusters of specified service count or more within a specified radius. Display number of services within each cluster.		
Stops: Show stops with specified number of transit services		
Network routes based on connection criteria	A geo-spatial database allows geographical operations required for this type of analysis easier and more efficient to implement.	
Spatial connection problem areas	As discussed above.	
TEMPORAL CONNECTIVITY		
Map Display: Temporal connection problems	As discussed above.	
Combinations: Ability to constrain report by mixing limits across report Category	Need more clarification: which reports category and constrained are needed to be mixed.	