New York University Shanghai

CSCI-SHU 370-001

Object Oriented Programming

BART Interactive Visualization Report

Authors:
Kelvin Liu
Kenny Song

Emails:
kelvin.liu@nyu.edu
kenny.song@nyu.edu

May 12, 2015

Contents

1 Overview

Our project is an interactive visualization built in Java for the rapid transit system of San Francisco called BART, or "Bay Area Rapid Transit". This public transit system consists primarily of a heavy-rail train and subway system that connects San Francisco with surrounding regions, including the East Bay area and northern San Mateo county. BART is claimed to be the fifth-busiest heavy rail rapid transit system in the US, having served on average 422,490 weekday passengers, 211,288 Saturday passengers, and 158,855 Sunday passengers in September 2014.

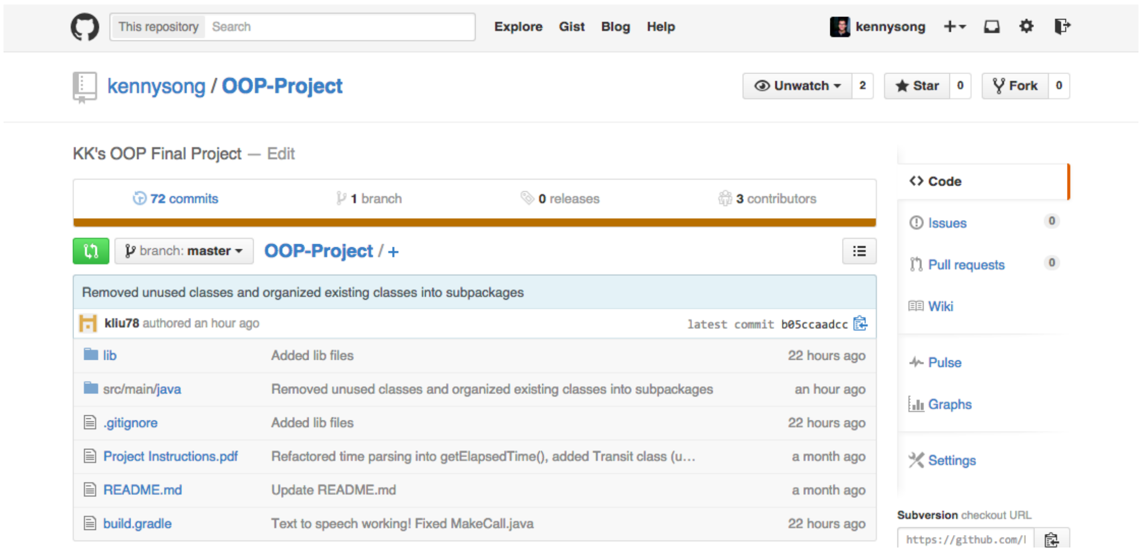
Being located in the heart of Silicon Valley, BART releases several data sets and open APIs for developers to create applications to help improve the passenger experience and also provide useful analyses and tools for administration purposes. See http://api.bart.gov/docs/overview/index.aspx. For our project, we decided to utilize two BART APIs (along with several other Java packages, frameworks, and APIs) to create an interactive visualization of the BART system. The two BART APIs we utilized were the GTFS data feed, as well as the real-time feed for their trains.

The motivation behind our project was to incorporate additional features that target accessibility purposes through non-visual methods of conveying transit data. First, as detailed in the project specifications, we created a Java application that interfaces with Google Maps to display train routes and real-time train location data. This was done using both JavaFX and the open source GMapsFX package, which provides Java bindings for Google Maps Javascript functionality.

Additionally, we utilized the open source FreeTTS Java package as a text-to-speech generator, which can read out arbitrary English text, and allows our application to speak to the user about the train visualization, such as "Line 02SFO10 has just arrived at the Balboa Park stop at 10:05am". Moreover, we utilized the public Twilio API with a free trial account to be able to both make calls and send SMSs to arbitrary phone numbers. This allows our application to send bilingual (English and Chinese) text messages and phone calls to a user at a specified time before a train arrives at their home stop.

This report will detail our code architecture, explore and discuss the various packages and APIs utilized along with their implementation choices, recall some difficulties and solutions we encountered, and propose several future steps for this project.

2 Code Architecture



kennysong / OOP-Project

KK's OOP Final Project — Edit

72 commits · 1 branch · 0 releases · 3 contributors

branch: master · OOP-Project / +

Removed unused classes and organized existing classes into subpackages

kliu78 authored an hour ago · latest commit b05ccaadcc

lib	Added lib files	22 hours ago
src/main/java	Removed unused classes and organized existing classes into subpackages	an hour ago
.gitignore	Added lib files	22 hours ago
Project Instructions.pdf	Refactored time parsing into getElapsedTime(), added Transit class (u…	a month ago
README.md	Update README.md	a month ago
build.gradle	Text to speech working! Fixed MakeCall.java	22 hours ago

We hosted all of our project code on a public repository on Github in order to effectively collaborate on this programming assignment. The use of git also allows us to access previous versions of our code and review changes, both of which have been useful features in developing this project. Our repository can be found by going to `https://github.com/kennysong/OOP-Project`.

In order to maintain organization for our project, we used Gradle, see `https://gradle.org/`. Similar to older tools such as Apache Ant and Apache Maven, Gradle is a build automation and dependency management tool. However, Gradle has multiple benefits over these tools. It is open source, actively under development, extensible via user-defined tasks and plugins, and supports wide-variety of languages in addition to Java (e.g. Groovy, Scala, C++, JavaScipt, etc.).

For any given Gradle project, the file `build.gradle` is essential. This file details the project's plugins, configuration, organization of source code, and external dependencies and is wholly customizable.

With the Java plugin applied, a developer gets access to a number of Gradle tasks, which simplify the development process of a java application. For example, if a developer runs the command `gradle build` in the root project directory, Gradle will not only complie the developer's code, but also run any defined test classes as well as package an executable `.jar` file containing compiled code. For this project in particular, Gradle was indispensable, as it allowed us to separate our source files from compiled class files, quickly declare external dependencies, and very easily run our application without having to deal with setting the class path ourselves.

The code we wrote is contained in the package, `oop.project`. Below is the hierarchy of this package and a high-level explanation of its contents:

```
oop.project
 ├─ bart_gtfs
 ├─ MapApp.java
 ├─ model
 │   ├─ Coordinate.java
 │   ├─ Stop.java
 │   └─ Trajectory.java
 ├─ util
 │   ├─ GTFSParser.java
 │   ├─ MakeCall.java
 │   ├─ SMSSender.java
 │   └─ TextToSpeech.java
 └─ view
     ├─ root.fxml
     ├─ sidebar.fxml
     └─ SidebarController.java
```

`bart_gtfs` is a directory that contains GTFS data provided by BART. This data is parsed by the application in order to load the trajectory of each train and each BART stop, separated by route. It is used in combination with the GTFS-realtime feed, which is explained in close detail in section 3.5.

`oop.project.MapApp` is the main class of this application. It creates and displays the GUI with which the end user interacts. A basic overview of this class is illustrated by the method calls found in the `start` method. The application begins by loading the GTFS data needed on separate threads. It also creates JavaFX services to continually check the realtime feed and update the screen. Next, visual elements are created. The sidebar is loaded from an FXML file and the main map is created with GMapsFX. JavaFX and GMapsFX are explained in better detail in sections 3.1 and 3.2, respectively.

`oop.project.model` is a sub-package that contains model objects used in this application:

`oop.project.model.Coordinate` is a model for latitude-longitude coordinates. It stores latitude and longitude as two separate fields.

`oop.project.model.Stop` is a representation of a BART stop. Among information about the stop itself and the route it belongs to, it contains a Coordinate object, which represents the location of the stop.

`oop.project.model.Trajectory` describes a single BART trip. It also contains extrapolated information about the location of a train, given a certain time. This is stored in `SortedMap`.

`oop.project.util` is a sub-package that contains utility classes used in this application:

`oop.project.util.GTFSParser` is used to parse the GTFS data.

`oop.project.util.MakeCall` is used to call the end user via the Twilio API. Twilio is explained in finer detail in section 3.3.

`oop.project.util.SMSSender` is used to send the end user SMS messages via the Twilio API. Twilio is explained in finer detail in section 3.3.

`oop.project.util.TextToSpeech` is used to render text as audio via FreeTTS. FreeTTS is further explained in section 3.4.

`oop.project.view` is a sub-package that contains FXML resources and view controllers:

`root.fxml` is a FXML file that describes an empty `BorderPane`. It has no associated controller class.

`sidebar.fxml` is a FXML file that describes the sidebar. It contains JavaFX Components that allow the end user to select how the application will notify about a train arrival.

`oop.project.view.SidebarController` is the controller class for `sidebar.fxml`. It allows us to programmatically access the JavaFX components inside `sidebar.fxml`.

3 Features

In this section, we discuss the external libraries, frameworks, and APIs utilized in order to create the various features of our interactive BART visualization. There are five main external resources that we used: JavaFX, GMapsFX, Twilio, FreeTTS, and the BART GTFS and real time data API.

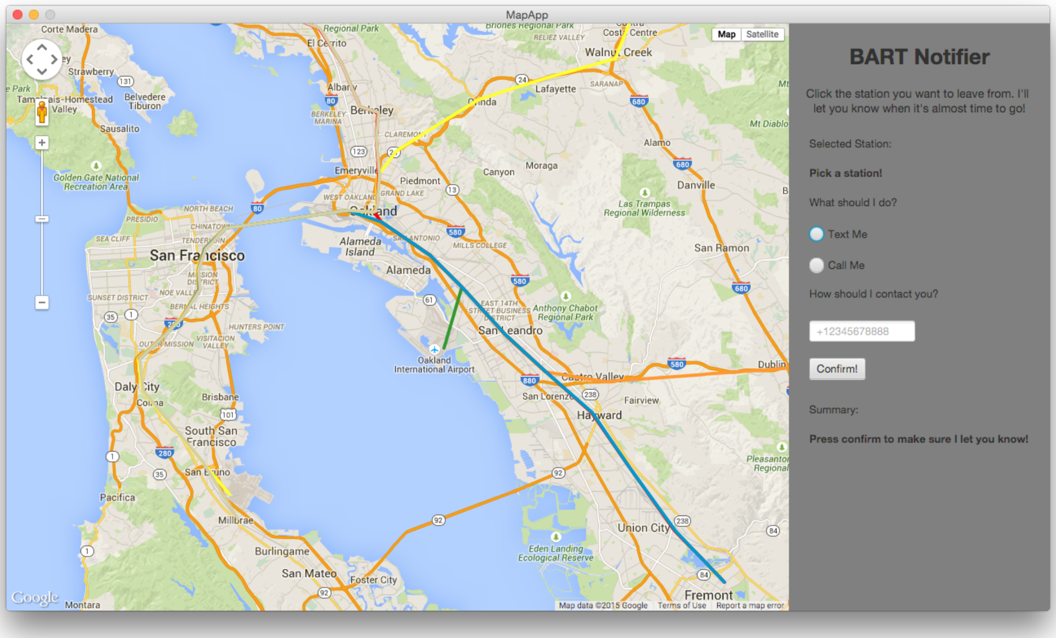
3.1 JavaFX

JavaFX is a native Java SE library that allows developers to create complex visual applications. It runs on all major desktop platforms (i.e. Windows, OS X, and Linux) as well as on some mobile devices and the web. JavaFX is intended to eventually replace the older Java GUI framework, Swing. In order to ease this transition and maintain backwards compatibility, JavaFX is able to extend existing Swing applications.

JavaFX has many features, which proved beneficial for this project. First, UI design is very flexible in JavaFX. Layout managers and components can either be programmed in a manner similar that of Swing or be defined in FXML files. The FXML files separate from the Java code. Furthermore, these FXML files can be generated by tools such as the Java Scene Builder, see `http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html`. Our project leverages this unique ability to create layouts separately from code. Additionally, each FXML file can be linked to a controller class, which provides access to the components defined in FXML.

Additionally, multithreading was greatly simplified using JavaFX. We are able to execute code on separate threads periodically using the provided `ScheduledService` class. These services are handled by the JavaFX runtime and are safely canceled when the application is quitting.

Below is a screenshot of our application. Like all JavaFX applications, the main class of our project extends the abstract class, `javafx.application.Application`. By doing so, we allow the JavaFX runtime to control the application's life-cycle. The application runs indefinitely, until explicitly closed by the user.



BART Notifier

Click the station you want to leave from. I'll let you know when it's almost time to go!

Selected Station:

Pick a station!

What should I do?

Text Me

Call Me

How should I contact you?

+12345678888

Confirm!

Summary:

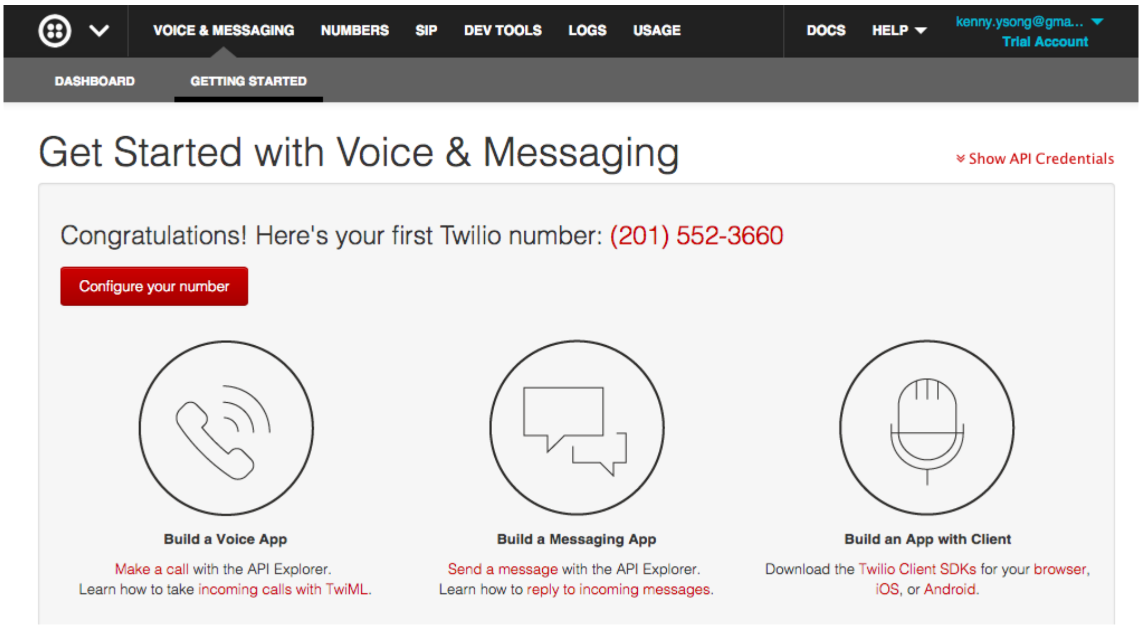
Press confirm to make sure I let you know!

3.2 GMapsFX

GMapsFX is an open source library that provides a Java API for Google Maps. Ordinarily, extensions and applications that rely on Google Maps are written in JavaScript. However, GMapsFX utilizes JavaFX's ability to render web content to create Java bindings for the Google Maps JavaScript API.

At the moment, GMapsFX does not provide full functionality with the Google Maps JavaScript API, but it is under active development. For the purposes of our project, GMapsFX is more than enough. It provides the end user with a familiar map interface. We have drawn colored lines representing different BART routes. Finally, a somewhat accurate location of any given train is displayed as a marker. We decided to use markers instead of drawing shapes to represent a train primarily because continually having to render multiple shapes turned out to be very taxing on system resources, while markers did not.

3.3 Twilio



Get Started with Voice & Messaging

Congratulations! Here's your first Twilio number: (201) 552-3660

Make a call with the API Explorer.
Learn how to take incoming calls with TwiML.

Send a message with the API Explorer.
Learn how to reply to incoming messages.

Download the Twilio Client SDKs for your browser, iOS, or Android.

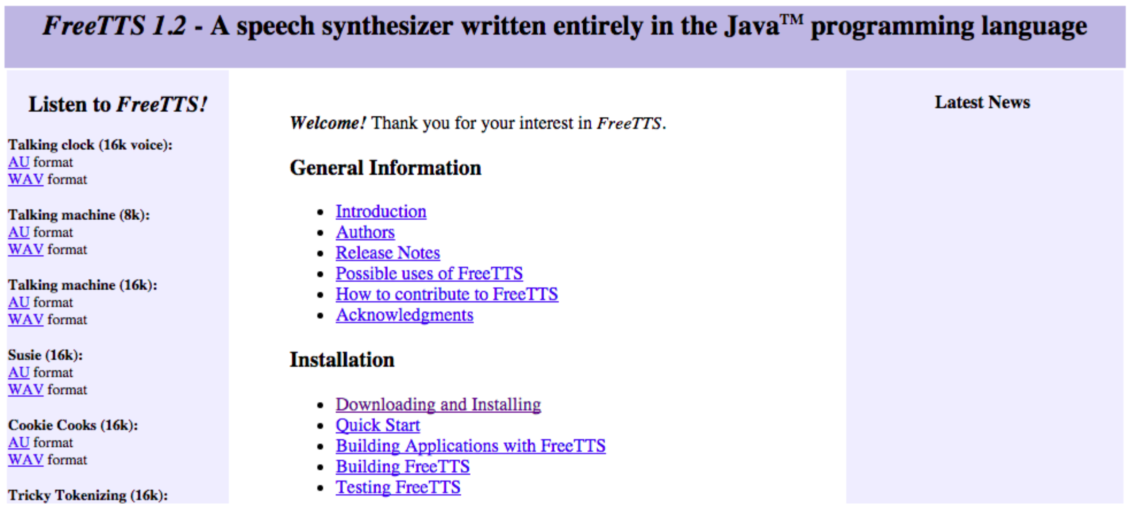
Twilio, see https://www.twilio.com/, is an API company that lets developers programmatically send and receive both text messages and calls. This API is enormously popular at hackathons, since the company has a great developer relations team and sponsors many of these events every year. At one point, we found a problem with their API, which they helped us debug in less than 10 minutes over email. They offer custom libraries in Ruby, PHP, Python, Node.js, C#, and Java. They also have a REST endpoint that any language can interface with directly through HTTP POST requests.

Essentially, we made a free developer account on their website, which allocated us one custom Twilio number that acts as our primary phone number (as seen in the screenshot above), and registered one personal phone number with them, which could be used to send unlimited calls and text messages for free. Contacting arbitrary numbers costs around \$0.015 per minute per call, and \$0.0075 per text message. It is also possible to contact Chinese phone numbers, but is more expensive and unavailable to use as a development phone number, so we used a Google Voice US phone number to receive calls and SMSs for the purposes of developing this project and for the demo.

We wrote two custom Java classes that interface with the Twilio API, `MakeCall.java` and `SMSSender.java`. Both contain our authentication ID and tokens to use the service. As their names suggest, `MakeCall` is a class that contains the static method (for ease of use in other classes) `makeCall(String toNumber)` that will call a number to the designated phone number from our Twilio number, and `SendSMS` is a class that contains the static method `sendSMS(String toNumber, String message)` that sends a string as a text message to the designated phone number.

For calls, `MakeCall` also includes an instance variable that contains the link to a publicly accessible TwiML file, which is basically an XML file that outlines what the computer will say once a call is connected to a user. We use this to control the dialogue when our code makes a call. Our TwiML file is hosted on `www.twimlbin.com`, a free service designed for this purpose. It was surprisingly hard to get this to work, since GitHub itself does not serve files with the correct Content-Type header that Twilio requires, and the CDN service Rawgit serves the correct header, but does not accept the POST request (instead of GET) that Twilio makes.

3.4 FreeTTS



***FreeTTS 1.2* - A speech synthesizer written entirely in the Java™ programming language**

Listen to *FreeTTS!*

Talking clock (16k voice):
AU format
WAV format

Talking machine (8k):
AU format
WAV format

Talking machine (16k):
AU format
WAV format

Susie (16k):
AU format
WAV format

Cookie Cooks (16k):
AU format
WAV format

Tricky Tokenizing (16k):

Welcome! Thank you for your interest in *FreeTTS*.

General Information

- Introduction
- Authors
- Release Notes
- Possible uses of FreeTTS
- How to contribute to FreeTTS
- Acknowledgments

Installation

- Downloading and Installing
- Quick Start
- Building Applications with FreeTTS
- Building FreeTTS
- Testing FreeTTS

Latest News

FreeTTS, see `http://freetts.sourceforge.net/docs/index.php`, is an open source Java package for speech synthesis, which takes am English text string and outputs it as audio in a variety of voices. We chose this over the more recent Mary package for text-to-speech since there was more documentation on how to set up this package, and it seemed that Mary was more difficult to use as it required setting up a speech synthesis server alongside our code. FreeTTS essentially provides a set of standalone jars that could be directly included in our application as a class.

Our text to speech code is contained inside the eponymous class `TextToSpeech.java`. The code is quite simple, we simply import the packages `com.sun.speech.freetts.Voice` and `com.sun.speech.freetts.VoiceManager`, and directly utilize them in the static method `speak(String text)`, that will handle converting the string to audio, and then playing the audio through the computer's speakers. We use the default voice "Kevin" for our project. The quality of the synthesized audio is not great, but usable for our demonstration purposes as a proof of concept.

One of the difficulties we encountered was that since FreeTTS was relatively old (last updated in 2009), it was not easily accessible in any of the modern Java package managers' repositories, such as Gradle and Maven. There were two online repositories that we found that hosted this code, but neither of them were actually from the FreeTTS team itself, and the mroe updated one did not include all the jar files needed for all of the classes. To solve this, we ended up just downloading the jar files manually from FreeTTS's SourceForge page and placing them in a top-level `lib` directory, which gets manually imported in our `gradle.build` file with the lines following `compile files('lib/cmu_time_awb.jar')`.

3.5 GTFS and Real Time Data

In addition to providing static GTFS data, BART has a realtime feed for trip updates at `http://api.bart.gov/gtfsrt/tripupdate.aspx`. The feed is encoded in a binary format and the data itself is in Google's GTFS-Realtime format. We are able to interact with this GTFS-Realtime data by using bindings that Google provides. Each time a request is sent to the feed, the response is a list of `FeedEntity`s. Each `FeedEntity` contains the trip id of a certain train and the train's next expected stop. This data is eventually animated by our application.

4 Conclusion

Our BART interactive visualization not only provides a real-time, visual map of all the of the trains in the BART public transit system, but also incorporates several features that significantly improve the accessibility and practicality of the application. The Java application can read out current BART train information using a text-to-speech synthesizer. Additionally, when the user is away from his computer, he can still benefit from this application by receiving either calls or text messages to update him on his train statuses. This was accomplished through several external resources, including JavaFX, GMapsFX, Twilio, FreeTTS, and the BART GTFS and real time data API.

Many of the difficulties we encountered and our solutions are detailed in the pages above, and most of them stem from figuring out how to interface with these external packages and APIs. We used Gradle to effectively manage all of our project files and dependencies in an extensible and modern way, and also published our code on Github, see https://github.com/kennysong/OOP-Project, for easy collaboration with two people developing the project. We were able to successfully implement all the features we planned.

Looking forward, there are several steps that can take this project even further and provide more practical value for users. Currently, our call does not provide any custom information about the user's train, besides ”Your train is arriving soon”, so it could be useful to programmatically generate a new TwiML file that includes the custom information for that user's train, which Twilio can use during its call. Additionally, we could include more information and places where the text-to-speech engine is utilized, perhaps even in the menu, so that the application can be usable purely through audio.