Timelines

Design Document

CMPT371-Team2

Table Of Contents

Architecture

Working from a high level downwards; the architecture is Model-View-Controller as required by React. The view component is handled entirely by React. The controller will be handled by internally developed React components and a pipes-and-filters entry point into the system. Finally, the model is handled by a blackboard sub-architecture.

The pipes and filters design will be useful when the data first enters the system and is streamlined through the Parser class which handles all formatting, determining data types and creating the timeline. The blackboard design pattern is responsible for the core functionality of the system once the entering data has been handled. The blackboard design's strength is in decoupling the functionality and the memory.

The blackboard design pattern traditionally has three components: blackboard, knowledge source and control component. The control component is the Filter class as it determines which data is drawn and how. It accomplishes this by working in tandem with the Column and Draw classes. The blackboard component is the timeline class as it stores all of the current data. The Column class is the knowledge source as it stores the information about how the data in question will be represented.

The Parser handles all of the logic of a pipes and filters design pattern however it is possible that this class will be divided in the future if we determine the class cuts across too many concerns.

Strength of architecture:

Pipes-and-filters entry-point: Modularity of data import process. The pipes-and-filters encapsulates all of the logic of pre-processing the data while allowing for future concurrency extensionality. This will be useful as scale the system to allow for larger datasets.

Blackboard data-control: This was chosen due to a specific technical constraint in JavaScript where data cannot be passed by reference. Blackboard allows us to address the concern of efficiency in both memory management and speed because we're using a single global array for each dataset.

Main Classes:

Parser

The data handler class, acts as an entry point for data into the system. Responsible for for taking in a .csv file, sorting it, inferring the type of the data, and defining the allowable draw methods. Then responsible for instantiating a timeline, and populating it with Column classes.

In the future this class will include the functionality to upload a .tl (timeline) file and reconstructing a previous workspace.

Fields:

prompt: a string that's the prompt for the user

fileType: a filetype object, which is seen as acceptable by the parser

Methods:

constructor(props: ParserInterface):

isValid :: event → boolean

    Inputs: the js event passed in by the html file input

    Output: a boolean indicating success or failure

    Side Effects: array of data objects flagged as valid, or invalid

sortData :: Array<object> → boolean

    Inputs: array of data

    Output: a boolean indicating success or failure

    Side-effects: the array of data sorted by a time field

inferTypes :: Array<object> → Array<Column>

    Inputs: array of data

    Output: a list of objects which define the methods available for the data

    Side-effects: None

parse :: fileEvent → Array<Data>

    Inputs: the js event passed in by the html file input

Output: a CSVData object

Side-effects: None

parseCSV :: fileEvent → Array<Data>

Inputs: a .csv file

Output: an array of data objects

Side Effects: The creation of a timeline, it's associated columns

parseTL :: fileEvent → void

Inputs: a .tl file

Output: None

Side-effects: The workspace is fully instantiated

Flow of Parser Diagram
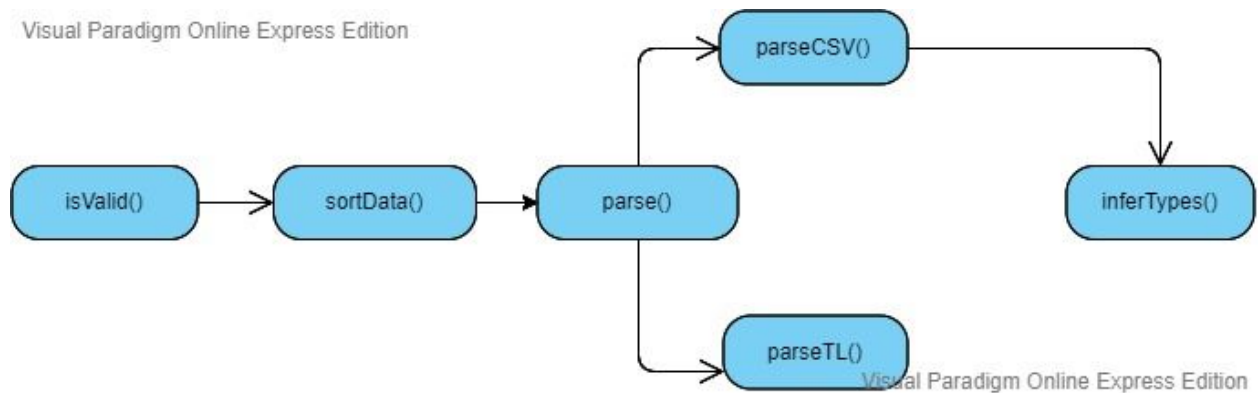
<u>Column</u>

The class responsible for a single column of data within a timeline.

Fields:

yScale: float which corresponds to the y scaling for data being drawn

key: a string descriptor of the column name which will be drawn

primType: a string descriptor of the type that column is associated with

definedDrawList: a list of draw objects which are defined for the type of data the column represents

Methods:

constructor (float: yScale, Array<Draw>: definedDrawList, string: key, string: primType)

show :: void → void

      Inputs: none

      Outputs: none

      Side-effects: Renders a column of data to the canvas

rescale :: float → void

      Inputs: a float which is the new scaling factor

      Outputs: none

      Side-effects: Causes the column to be rendered at a new scale

Data

Class responsible for actually storing the data from a .csv file.

Fields:

arrayOfData: an array of the actual data objects

filter: a filter object

pathToData: a string which stores the path to the .csv

currentXColumn: a string which is the label of the column currently being used by the active timeline as the X axis label

currentYColumn: a string which is the label of the column currently being used by the active timeline as the Y axis label

Methods:

constructor (string: pathToData, Array<Object>: arrayOfData, filter: filter)

getData :: void → Array<Object>

      Inputs: none

      Outputs: the arrayOfData

      Side-effects: none

setData :: Array<Object> → void

      Inputs: an array of data to be stored by the object

      Outputs: none

      Side-effects: none

getFilter :: void → filter

      Inputs: none

      Outputs: A reference to the filter object for a dataset

      Side-effects: none

setPath :: string → void

      Inputs: A string, which is a fully validated path name to where the data sits on the users computer.

      Outputs: None

      Side-effects: None

getPath :: void → string

      Inputs: None

      Outputs: A string, which is a fully validated path name to where the data sits on the users computer.

      Side-effects: None

Draw

Class responsible for storing the actual draw methods that column classes will use as rules to render themselves. This is really just a wrapper for a function, and the function name.

Fields:

drawFunction: a reference to a function that the column uses to actually render itself

functionDesc: a string which is the 'pretty printed' version of the function name, for rendering in dropdown menus.

Methods:

constructor (function(any → void) : drawFunction, string: functionDesc)

getDrawName :: void → string

      Inputs: none

      Outputs: returns a descriptive string for the draw function that the object is associated with

      Side-effects: none

getDrawFunction :: void → function(any → void)

      Inputs: none

Outputs: returns a reference to the drawFunction function

Side-effects: none

Predicate

Wrapper class for predicates to be used by the filter class. This holds a reference to a string, and a well-formed predicate (a function which returns true or false for a singular element).

Fields:

name: string

predicate: function: any → boolean

Methods:

constructor (string: name, function(any → boolean))

getName :: void → string

Inputs: None

Outputs: A string, the name of the predicate

Side-effects: None

Filter

Class responsible for storing the information about which data is drawn. By default it stores only information about the range that the data is drawn in. Optionally, it also filters the data by predicates, allowing the user to define things like "only data less than 5" or something similar.

Fields:

lowerRange: integer

upperRange: integer

isShown: a boolean which determines if a given column is actually being drawn

listOfPreds: a list of predicate objects

Methods:

constructor(integer: lowerRange, integer: upperRange, boolean: isShown)

redefineRange :: (integer, integer) → void

Inputs: new lower_range and upper_range

Outputs: none

Side-effects: the lower_range, and upper_range values are reassigned

addPredicate :: predicate → void

Inputs: a well formed, named predicate which relates to the data

Outputs: none

Side-effects: the new predicate is appended to the list of predicates

removePredicate :: string → void

Inputs: the name of a specific predicate to remove

Outputs: none

Side-effects: the indexed predicate is removed from the list

getAllPreds :: void → Array<predicate>

Inputs: none

Outputs: listOfPreds, a list of key value pairs

Side-effects: none

toggleFilter :: filter → void

Inputs: the filter being toggled

Outputs: none

Side-effects: Sets isShown to *not* isShown

Timeline

Container class for columns. Its draw method simply calls the draw method of all of the columns associated with it.

Fields:

listOfColumns: list of objects of type Column class

isShown: a boolean. True if data is shown.

bigDataArray: array of the input data from the .csv

Methods:

constructor(Array<Column> : listOfColumns)

The constructor for the class takes in an array of all of the data. It then uses the data to populate its own internal list of columns, and by default sets its own 'isShown' variable to true.

show :: void → void

       Inputs: none

       Outputs: none

       Side-effects: calls the show method on all of the Columns in listOfColumns

toggleVisibility :: void → void

       Inputs: none

       Outputs: none

       Side-effects: sets isShown to *not* isShown.

Top Level Functions

This part of the document outlines the main, purely functional components of the program. These functions were included to support a pseudo functional style, where each timeline only has access to it's defined functions, that way it's easier to reason about correctness when debugging any rendering aspects of the code.

drawIntervalData :: (Data, Filter, float) → void

A function which takes in a data set, and draws it to the canvas as Interval Data.

Pre-Conditions:

A data set exists which can be drawn as either interval data, or quantified interval data.

Post-Conditions:

The data set is drawn, as interval data, to the canvas.

Parameters:

data: A reference to the data set to be drawn

filter: A reference to the filter which determines which data will be drawn

yScale: A float which represents the scale at which the data is to be drawn.

drawQuantifiedIntervalData :: (Data, Filter, float) → void

A function which takes in a data set, and draws it to the canvas as as quantified interval data.

Pre-Conditions:

A data set exists which can be drawn as quantified interval data

Post-Conditions:

The data set is drawn, as interval data, to the canvas.

Parameters:

data: A reference to the data set to be drawn

filter: A reference to the filter which determines which data will be drawn

yScale: A float which represents the scale at which the data is to be drawn.

drawPointData :: (Data, Filter, float) → void

A function which takes in a data set, and draws it to the canvas as a series of points data.

Pre-Conditions:

A data set exists which can be drawn as quantified interval data

Post-Conditions:

The data set is drawn, as interval data, to the canvas.

Parameters:

data: A reference to the data set to be drawn

filter: A reference to the filter which determines which data will be drawn

yScale: A float which represents the scale at which the data is to be drawn.

drawQuantifiedPointData :: (Data, Filter, float, boolean) → void

A function which takes in a data set, and draws it to the canvas as as quantified point data.

Pre-Conditions:

A data set exists which can be drawn as quantified interval data

Post-Conditions:

The data set is drawn, as interval data, to the canvas.

Parameters:

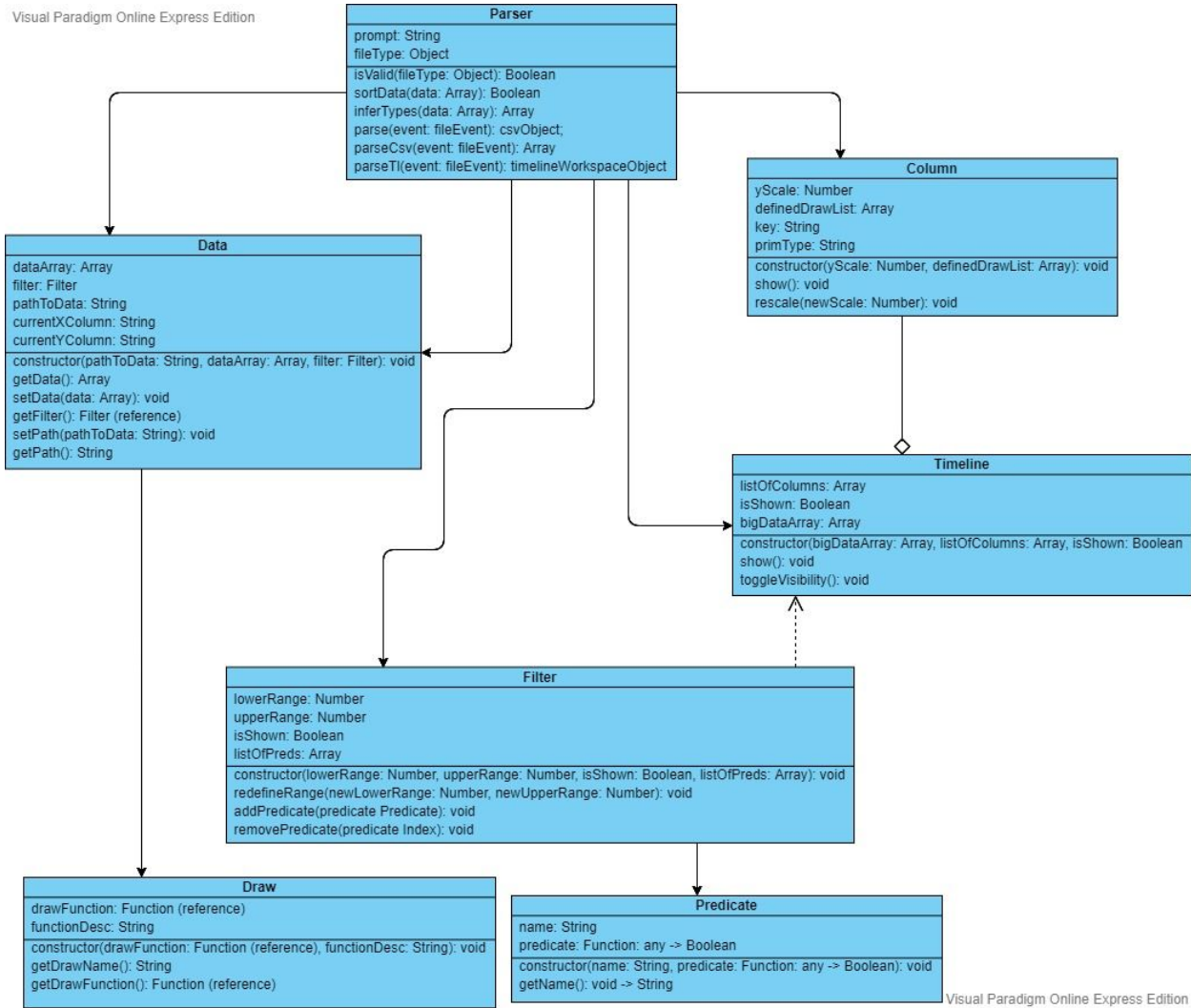**data**: A reference to the data set to be drawn

filter: A reference to the filter which determines which data will be drawn

yScale: A float which represents the scale at which the data is to be drawn.

connected: A boolean which specifies whether the data is to be connected with piecewise lines or not.

UML Diagram

Parser

prompt: String
fileType: Object

isValid(fileType: Object): Boolean
sortData(data: Array): Boolean
inferTypes(data: Array): Array
parse(event: fileEvent): csvObject;
parseCsv(event: fileEvent): Array
parseTl(event: fileEvent): timelineWorkspaceObject

Column

yScale: Number
definedDrawList: Array
key: String
primType: String

constructor(yScale: Number, definedDrawList: Array): void
show(): void
rescale(newScale: Number): void

Data

dataArray: Array
filter: Filter
pathToData: String
currentXColumn: String
currentYColumn: String

constructor(pathToData: String, dataArray: Array, filter: Filter): void
getData(): Array
setData(data: Array): void
getFilter(): Filter (reference)
setPath(pathToData: String): void
getPath(): String

Timeline

listOfColumns: Array
isShown: Boolean
bigDataArray: Array

constructor(bigDataArray: Array, listOfColumns: Array, isShown: Boolean
show(): void
toggleVisibility(): void

Filter

lowerRange: Number
upperRange: Number
isShown: Boolean
listOfPreds: Array

constructor(lowerRange: Number, upperRange: Number, isShown: Boolean, listOfPreds: Array): void
redefineRange(newLowerRange: Number, newUpperRange: Number): void
addPredicate(predicate Predicate): void
removePredicate(predicate Index): void

Draw

drawFunction: Function (reference)
functionDesc: String

constructor(drawFunction: Function (reference), functionDesc: String): void
getDrawName(): String
getDrawFunction(): Function (reference)

Predicate

name: String
predicate: Function: any -> Boolean

constructor(name: String, predicate: Function: any -> Boolean): void
getName(): void -> String