Timelines

Design Document

CMPT371-Team2

Table Of Contents

1. Architecture
2. Main Classes
    a. Parser
        i. Parser Diagram
    b. Column
    c. Data
    d. Draw
    e. Predicate
    f. Filter
    g. Timeline

3. Top Level Functions

    h. drawQuantifiedIntervalData
    i. drawIntervalData
    j. drawPointData
    k. drawPointData

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.

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. It is responsible 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):

createNewMockFileEvent :: void →any

Pre-conditions: a file with valid file data and a valid file name and a valid file type

Post-conditions: None

Inputs: None

Outputs: A mock file event similar to the actual file event

Side-effects: None

checkifCsvandcallParse:: void→boolean

Pre-conditions: the current file should have a valid name and type

Post-conditions: parse is called if file is .csv

Inputs: None

Outputs: boolean indicating if parse was called

Side-effects: parse method is called if the requirements are met

dataIsValid

Pre-conditions: a csv has been uploaded and its data is stored in the data array state

Post-conditions: None

Inputs: array of data

Outputs: return true if data is valid and throw error if nothing exists

Side-effects: None

lookForDateKey

Pre-conditions: a csv has been uploaded and its data is stored in an array

Post-conditions: None

Inputs: array of data

Outputs: the key of the first valid date format found

Side-effects: None

isValid :: event → boolean

Pre-conditions: no other parser object exists

Post-conditions: a parser object is instantiated as the only parser object

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

Pre-conditions: a csv has been uploaded, and the data is stored in an array

Post-conditions: the data stored in the array is sorted by some date column

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 | Undefined

Pre-conditions: An array of sorted data exists for types to be inferred from

Post-conditions: The array is transformed into an array of type Column, and the default behavior for the data is inferred and set

Inputs: array of data

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

Side-effects: None

parse :: fileEvent → Array<Data>

Pre-conditions: The user uploads a file

Post-conditions: The data is read out of the csv file and put into an array object

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

Output: a CSVData object

Side-effects: None

parseCSV :: fileEvent → Array<Data>

Pre-conditions: The user has uploaded a csv file

Post-conditions: The data from the file is stored in a global array

Inputs: a .csv file

Output: an array of data objects

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

render :: void

Pre-conditions: None

Post-conditions:

Inputs: None

Outputs: None

Side-effects: Renders html upload component to the canvas

createTypeCountingObjects :: int → List<CountTypes>

Pre-conditions: None

Post-conditions: None

Inputs: fieldLength: the number of columns of data

Outputs: A list of columns

Side-effects:

inferTypesHelper:: string[] → CountTypes[]

Pre-conditions: argument is a non-empty list

Post-conditions: creates a non-empty list of CountTypes

Inputs: fieldList: the list of fields for each column

Outputs: an array of C

createColumn :: string, enumDrawType, number, string[], Column[] → void

Pre-conditions:

Post-conditions:

Inputs: fieldList: the list of fields for each column

Outputs: an array of Columns

CountTypes

This internal class helps counting of types of data in a single Column

Fields:

numNumber: The number of ‘number’ type elements in a column

numIncongruent: The number of ‘incongruent’ type elements in a column

numString: The number of ‘string’ type elements in a column

numData: The number of ‘date’ type elements in a column

Methods:

largest :: void → string

Finds the largest element of the fields. (aka the type of data that is most common in the column)

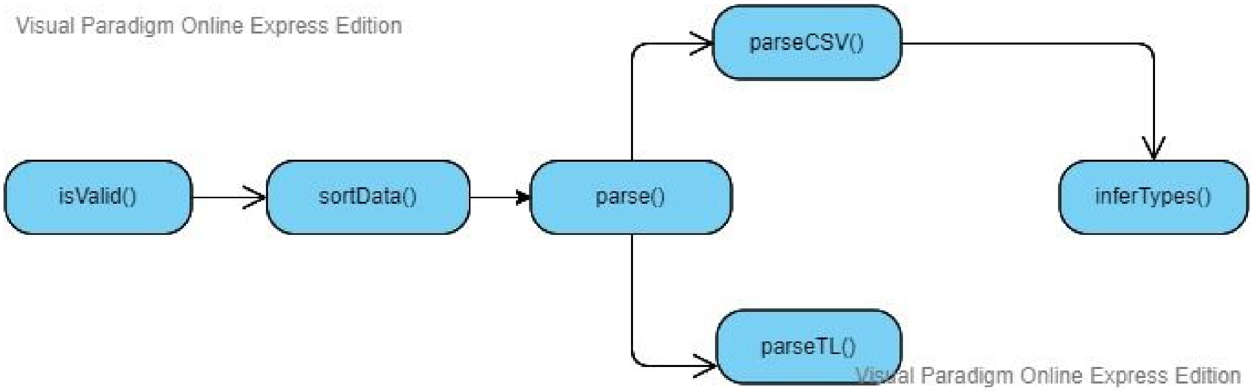
Pre-conditions: None

Post-conditions: None

Inputs: None

Outputs: A string representing the largest field. Defaults to ‘date’ if all are zero.

Flow of Parser Diagram


Visual Paradigm Online Express Edition
parseCSV()
isValid()
sortData()
parse()
inferTypes()
parseTL()
Visual Paradigm Online Express Edition

Column

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

Fields:

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

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

Methods:

constructor (float: yScale, enumDrawType: drawType, string: key, string: primType)

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

columns: an array of columns

Methods:

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

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

Pre-conditions: The predicate name is set

Post-conditions: The predicate name is returned

Inputs: None

Outputs: A string, the name of the predicate

Side-effects: None

evaluate :: Object → bool

Pre-conditions: The predicate is defined, and the object being passed in is the correct type

Post-conditions: The predicate is evaluated with the passed in object and true or false returned depending on whether or not the predicate holds for that object

Inputs: The data point to evaluate if the predicate holds

Outputs: The result of the predicate function

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, Array<any → boolean>: listOfPreds)

redefineRange :: (integer, integer) → void

Pre-conditions: None

Post-conditions: A filter object is created

Inputs: new lower_range and upper_range

Outputs: none

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

addPredicate :: predicate → void

Pre-conditions: None

Post-conditions: The predicate is added to the list of predicates and this is immediately applied to the current render

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

Pre-conditions: The predicate list is non-empty

Post-conditions: The specified predicate is removed from the list. If no such predicate exists nothing is removed.

Inputs: the name of a specific predicate to remove

Outputs: none

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

TimelineModel

This class is responsible for holding various values that are used to render the timeline component. The constructor for this class just sets up the model with default values for all of its fields

Fields:

marginTop: integer : margin from the top of the browser

marginBottom: integer : margin from the bottom of the browser

marginLeft: integer : margin from the left side of the browser

marginRight: integer : margin from the right side of the browser

scaleZoomOut: float : how much the scaling factor decreases on a zoom out

scaleZoomIn: float : how much the scaling factor increases on a zoom in

deltaPan: integer : how much panning is done when using the arrow keys to pan

scaleMin: float : the minimum scale value

fullWidth: integer : the full width of the browser (margins are subtracted from this for render area)

fullHeight: integer: the full height of the browser (margins are subtracted from this for render area)

Height: integer : the margin adjusted height of the render area

Width: integer : the margin adjusted width of the render area

barWidth: integer : the width of bars when rendered

barBuffer: integer : the number of bars to keep in the buffer for rendering

numBars: integer : the numbers of bars in total

dataIdx: integer: the index of where the data is being drawn

deltaX: integer : the amount that the screen pans when using the arrow keys xColumn: string : the name of the first x column

xColumn2: string : the name of the second x column

yColumns: List<Column> : a list of y columns

xColumns: List<Column> : a list of columns

csvData: List<Object> : a list of all of the data loaded in from the uploaded csv

Data: Array<object> : an array of all of the data

minDate: any : the minimum date on the timeline

maxDate: any : the maximum date on the timeline

timeScale: any : the scaling for the x axis

View: ViewType : a reference to a enum, which is the current drawing style for the timeline

TimelineComponent

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

Fields:

state: A wrapper for the current state of the timeline component. Has information about margins, height and width for rendering, and holds a reference to the data to render.

Methods:

constructor(

> **data :** a reference to the data that will be rendered
>
> **Width :** the width that the timeline will be rendered at
>
> **Height :** the height that the timeline will be rendered at
>
> **marginTop :** the margin from the top of the browser
>
> **marginBottom :** the margin from the bottom of the browser
>
> **marginLeft :** the margin from the left side of the browser
>
> **marginRight :** the margin from the right side of the browser
>
> **yColumn :** name of the selected y column
>
> **xColumn :** name of the selected x column

xColumn2 : name of the other selected x column

Loading : boolean that keeps track of whether the file is being loaded or not

View : reference to an enum which reflects how the data is being drawn

)

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.

initTimeline :: void → void

Pre-conditions: data has been read in and parsed, and exists in this.state.data

Post-conditions: the initial values for rendering set

drawTimeline :: void → void

Pre-conditions: initTimeline has been called, and there exists an instance of the timeline class to draw

Post-conditions: the timeline is rendered to the canvas

drawLabels :: void → void

Pre-conditions: Labels exist to be rendered

Post-conditions: Labels are rendered to the canvas

registerEvents :: void → void

Pre-conditions: An keyboard/mouse event occurs

Post-conditions: The correct event is registered

ttOver :: any → void

Pre-conditions: An element exists in the timeline that can have a tool tip rendered

Post-conditions: The tool tip is rendered on the canvas

ttUpdatePos :: (number, number) → void

Pre-conditions: A tooltip is currently being rendered

Post-conditions: The tooltip has it’s x and y position updated to follow the mouse

ttMove :: any → void

Pre-conditions: A valid datum is passed into the function from the timeline

Post-conditions: ttUpdatePos is called and the component is rendered

ttLeave :: any → void

Pre-conditions: None

Post-conditions: Any tooltips that exist are removed from the list of currently drawn tooltips

updateChart :: void →void

Pre-conditions: An event occured that caused update chart to be called (d3.event is not null)

Post-conditions: The correct event is triggered which updates the chart

updateBars :: void → void

Pre-conditions: A timeline type exists

Post-conditions: The bars are updated

moveChart:: void → void

Pre-conditions: A timeline type is instantiated

Post-conditions: The chart position is updated with the new values

dragged :: void → void

Pre-conditions: A timeline is instantiated and being rendered

A mouse click occured

Post-conditions: The chart position is updated

dragStarted :: any → void

Pre-conditions: A timeline object exists

Post-conditions: The drag state is set is to true

dragEnded :: any → void

Pre-conditions: The drag state is set to true

Post-conditions: The drag state is set to false and the dragging stops

changeTimelineType :: any → void

Pre-conditions: A csv file has been loaded

A timeline model class has been instantiated

Post-conditions: The selected timeline type has changed

Render :: void → string

Pre-conditions: None

Post-conditions: None

resetTimeline :: void → void

Pre-conditions: A timeline component exists and is being rendered

Post-conditions: A timeline component is re-instantiated

changeColumn :: any, string → void

Pre-conditions: At least one other column is instantiated to draw

Post-conditions: The currently rendered column is changed

sortData :: string → void

Pre-conditions: The string passed in is present as the header for one of the columns

The column associated with that string can be ordered

Post-conditions: The data is sorted by the column associated with the string passed in

componentDidMount :: void → void

Pre-conditions: the component mounted correctly

Post-conditions: default x and y columns are chosen

getDeltaX :: void → number

Pre-conditions: A timeline model is instantiated

Post-conditions: None

getScale :: void →number

Pre-conditions: A timeline model is instantiated

Post-conditions:None

mapColumnsToOptions:: array → JSX.Element

Pre-conditions: none

Post-conditions: none

ttOverHelper:: (any,number,number) → void

Pre-condition: input variables are not null

Post-condition: none

drawEventMagnitude ::any →void

Pre-condition: input variables are not null

Post-condition: none

drawIntervalMagnitude ::any →void

Pre-condition: input variables are not null

Post-condition: none

getEventMagnitudeData ::void →void

Pre-condition: this.timelineType.getData() is not null

Post-condition: none

getIntervalMagnitudeData ::void →void

Pre-condition: this.timelineType.getData() is not null

Post-condition: none

TimelineType

A class that is extended by the four main drawing classes, allowing them to share any common methods

Methods:

constructor(

newModel : a reference to an instance of the timeline model class

)

applyZoom :: any → void

Pre-conditions: A timeline is being rendered

Post-conditions: The zoom factor is changed if possible

draw :: (any, any, any, any) → void

Pre-conditions: Event elements exist to be rendered

Post-conditions: The selected components are drawn, any tooltips are also drawn

drawLabels :: any → void

Pre-conditions: A timeline is instantiated and has valid tags to be rendered

Post-conditions: Labels are drawn for every point of data being rendered

outsideLeftBound :: [number] → boolean

Pre-conditions: the timeline has been correctly initialized with an appropriate value for deltaX and width

Post-conditions: A boolean is returned representing if the values are within the bounds.

outsideRightBound :: [number] → boolean

Pre-conditions: the timeline has been correctly initialized with an appropriate value for deltaX and width

Post-conditions: A boolean is returned representing if the values are within the bounds.

getData :: void → void

Pre-conditions: the timeline component has correctly mounted and has a non-empty set of data

Post-conditions: the array of data to be rendered is updated to accurately reflect the information from the timeline

getTickTranslate :: any → string

Pre-conditions: A csv has been parsed and sent to the timeline component

Post-conditions: The bars and x-axis ticks are drawn correctly

updateBars :: (any, any, any) → void

Pre-conditions: A csv has been parsed and sent to the timeline component

Post-conditions: The bars and x-axis ticks are drawn correctly

checkYPrimType :: string → boolean

Pre-conditions: the primType accurately represents one of the columns from the parsed csv.

Post-conditions: True or false, based on whether or not the primType is valid for the timeline type.

checkXPrimType :: string → boolean

Pre-conditions: the primType accurately represents one of the columns from the parsed csv.

Post-conditions: True or false, based on whether or not the primType is a date or number.

updateColumns :: void → void

Pre-conditions: A timeline component exists and has data that can be rendered

Post-conditions: The correct drawing method is set