Scientific Experiment Browser

Stakeholders: Daniel Garijo, Yolanda Gil

Deliverable #7: Documentation

April 10, 2017

Huy Ngo
Andrew Nguyen
Tiffany Truong

Table of Contents

Setting Up

Viewing the project

The project's repository can be viewed at https://github.com/tetruong/Portal2.0. To view the portal itself, click the "View Portal" link in the README file.

Setting up project

1. git clone https://github.com/tetruong/Portal2.0.git
2. Setting up react (once you navigate to the project directory)
    a. npm install -g babel
    b. npm install -g babel-cli
    c. npm install --save react
    d. npm install --save react-dom
    e. npm install --save-dev babel-core babel-loader babel-preset-react babel-preset-es2015
    f. npm install --save-dev webpack webpack-dev-server html-webpack-plugin
3. If making changes to .jsx files, run command "./node_modules/.bin/webpack -d --watch" to recognize changes and automatically recompile .jsx changes
    a. This will automatically update bundle.js, bundle.js.map
4. In our development, we used Brackets (http://brackets.io/)- this allows you to spin up a local instance of the website and test changes live.
    a. With Brackets, File -> Open Folder, choose the project folder which includes html/, css/, and js/.
    b. To launch from the landing screen, navigate to html/index.html, and click the lightning icon on the top right of the window to launch a browser window with the workflow explorer.
        i. When you navigate to workflow-main.html on Brackets, if the lightning icon is still highlighted, the browser will automatically update the browser with the new page. Any changes made code within Brackets will live update on the browser if the lightning icon is still highlighted.
        ii. Note that when you open up code inspector view within the browser, the lightning icon will no longer be highlighted and you will need to refresh the browser in order to see any code changes.
5. Note that d3.min.js, dagre-d3.min.js, are external .js files used for the visualization

Some React Specific Notes

To configure the project, look at 'package.json' and 'webpack.config.js'. In these files, you can manage dependencies, entry points to project, and add command-line scripts to aid with development. For example, in package.json, in the 'scripts' section, there is the 'start' script, which will spin up a local instance at localhost:portNumber, where portNumber is specified in webpack.config.js.

HTML Files

index.html

HTML file for home/landing page of the site. This is where one can search for workflows and proceed to view a workflow page.

workflow-main.html

HTML file for displaying a workflow. It has two tabs (Workflow and Execution) on which to view separate visualizations. Here, the user can explore nodes in the visualization by clicking on process/variable nodes and having panels display to the right of the visualization. On the Execution tab, the user can explore different execution traces via a selection menu as well as view the metadata associated with a given execution trace.

CSS Files

landing-page.css

CSS file that contains the CSS styles of index.html.

workflow-style.css

CSS file that contains the CSS styles of workflow-main.html

Javascript Files

bundle.js/bundle.js.map

Auto-generated and updated file by React's webpack.

main.js

This file serves as the entry point to the project, where it renders the Title and search-bar React components. The entry point of the project can be changed within webpack.config.js.

Title.jsx

This file simply creates the title of the landing page and adds content to the html element with id 'title' in index.html.

visualization.js

This file contains the logic required to render the visualization of a workflow or execution trace. Data visualization libraries called D3.js (https://github.com/d3/d3/wiki) and Dagre.js (dagre-d3, more specifically https://github.com/cpettitt/dagre-d3) are used.

The visualization is comprised of a series of nodes and edges. Nodes are a general term to represent all variables and processes while edges represent the connections between nodes. Each node object, when set in the visualization, corresponds to a specific index inside the visualization. When setting a node object, you can set its label, labelStyle, shape, style in order to customize the look of the node depending on its name and whether or not it is an process or variable (for more on setNode method, look at d3/dagre-d3 specific functions table). When setting edges, you can also style them to your liking.

Since nodes and edges are regular Javascript objects, you can also add key-value pairings to incorporate extra data about a node. For example, a node's URI can be very helpful if some data about that process/variable needs to be queried from a repository's endpoint. Thus, when calling setNode, in addition to specifying other metadata for a node, we also include a corresponding URI.

Helpful dagre-D3/D3 functions	
setNode	Input: 1. Integer id, a unique number to identify the node in a graph 2. An object containing a node's metadata Output: Object containing information about entire graph Example usage: graph.setNode(1, { label: 'nodeLabelExample', shape: 'ellipse', style: 'fill: #fff',

	somethingUseful: 'useful-string-to-include' });
setEdge	Input: 1. Integer id of the source node 2. Integer id of the destination node 3. An object containing a node's metadata Output: Object containing information about entire graph Example usage: graph.setEdge(1, 2, { style: 'stroke: #000; fill:none;', });
graphObject.nodes	Input: none Output: Array of integer node ids
graphObject.node	Input: integer id of a node Output: Object containing all metadata/details of a given node

Variables	
svg/svgGroup	D3 selections used to manipulate UI/UX of the graph
vis	The graph object- setNode and setEdge are member functions of this object
zoom	Used to setup zoom behavior (https://github.com/d3/d3-zoom)
processInputMapping	Object acting like hash table Keys: URI strings of processes Values: Array of URI strings of input variables
processOutputMapping	Object acting like hash table Keys: URI strings of processes Values: Array of URI strings of output variables
isVariableOfMapping	Object acting like hash table Keys: URI strings of variables that are used by processes as inputs Values: Array URI strings of processes that use the variable as a parameter

outputByMapping	Object acting like hash table Keys: URI strings of variables output by process Values: Array of URI strings of processes that output the variable
processNodeIndices	Object acting like hash table Keys: URI strings of processes in visualization Value: integer index of node in visualization
putNodeIndices	Object acting like hash table Keys: URI strings of variables in visualization Value: integer index of node in visualization
workflowURI	String of URI of workflow/execution trace being rendered
results	Object containing query response with URIs for processes/variables required for visualization

Functions	
renderVisualization	Input: 1. AJAX response ‘res’ 2. boolean ‘isTrace’ Output: none Encompasses the logic required to set the nodes and edges of visualization and display to an svg element in workflow-main.html
addInputProcess	Input: 1. String URI of process ‘processName’ 2. String URI of input ‘inputName’ Output: none Adds inputName to the list of inputs for a given processName.
addOutputProcess	Input: 1. String URI of process ‘processName’ 2. String URI of output ‘outputName’ Output: none Adds outputName to the list of inputs for a given processName.

mapInputToProcess	Input: 1. String URI of input ‘inputName’ 2. String URI of process ‘processName’ Output: none Adds processName to the list of processes that use variable inputName
mapOutputFromProcess	Input: 1. String URI of output ‘outputName’ 2. String URI of process ‘processName’ Output: none Adds processName to the list of processes that output variable outputName - We believe that an output can only correspond to 1 process exactly, so the implementation of this can change in the future to reflect that
mapNodesEdges	Input: 1. Graph object ‘graph’ Output: none Parse ‘results’ object to set nodes of graph. Adds data to processOutputMapping, processInputMapping, isVariableOfMapping, outputByMapping, putNodeIndices, and processNodeIndices.
setGraphEdges	Input: 1. Graph object ‘vis’ Output: none Sets graph edges based on node indices from putNodeIndices and processNodeIndices.
formatInputs	Input: 1. Graph object ‘vis’ 2. Function ‘callback’ Output: none Calls getInputs from query.js to retrieve the inputs of a workflow and restyles the nodes determined to be inputs. After restyling nodes, executes the ‘callback’ function

renderGraph	Input: 1. Graph object ‘vis’ Output: none Populates svg element in workflow-main.html with visualization. Sets up height, width, zoom feature, centers graph.
setupNodeOnClick	Input: 1. D3 selection object ‘svg’ 2. Graph object ‘vis’ Output: none Setup on-click listeners for each node to display necessary information panels
translateVisualization	Input: none Output: none Translates visualization when information panels display
addTraces	Input: Object ‘traces’ Output: none Populates execution trace options in selection menu and adds event listeners for when the selection option is changed between different execution traces.
highlightPuts	Input: Array of URIs of inputs/outputs ‘putsArray’ Output: none Highlights the inputs/outputs of a given process node
unhighlightAllPuts	Input: none Output: none Unhighlights the inputs/outputs of all processes
unhighlightPuts	Input: Array of URIs of inputs/outputs ‘putsArray’ Output: none Unhighlights the inputs/outputs of a given process node
stripNameFromURI	Input: string ‘uri’

	Output: substring of 'uri' containing label of URI Parses URI for the relevant label to display
addDimensions	Input: dagre-d3 renderer object 'render' Output:none Adds dimensions functionality for workflows by creating custom shapes for nodes (https://github.com/cpettitt/dagre-d3/blob/master/lib/shapes.js).

query.js

This file contains functions containing SPARQL queries and AJAX calls required to populate the workflow page's dynamic content. There is a main endpoint set to query from at the top of the file.

Functions	
populateSearchBar	Input: 1. Function 'handler' Output: none Queries for the suggestions for auto-complete feature
getInputs	Input: 1. String 'workflow' URI 2. Function 'handler' Queries for the inputs of a workflow so their nodes can be styled properly in visualization
getWorkflowData	Input: 1. String 'workflowURI' 2. Function 'handler' Queries for the data of each node/edge in visualization of a workflow
getExecutionID	Input: 1. String 'workflowURI' 2. Function 'handler' Queries for the execution IDs of a given workflow and calls addTraces to populate options for selection box of execution traces

getExecutionData	Input: 1. String ‘executionID’ 2. Function ‘handler’ Queries for the data of each node/edge in visualization of an execution
getExecutionMetadata	Input: 1. String ‘executionID’ 2. Function ‘handler’ Queries for the metadata of an execution (status, time started, ended, when execution account was created

change-tabs.js

This file contains the logic in how to handle when a user switches between the ‘Workflow’ and ‘Execution’ tab. When the Execution tab is selected, icons indicating an execution trace’s metadata and a selection box will show on top of a new visualization.

Functions	
setWorkflowMetadata	Input: 1. Object ‘res’ Output: none Upon receiving query results, populates the metadata of an execution

search-bar.jsx

This file contains the react component for the autocomplete search bar in the index.html page. The main feature of this react component is the autocomplete functionality. In query.js, there is an AJAX function (populateSearchBar) that queries the endpoint and fetches a json file of the workflow URIs and labels. When our react component is first mounted, it calls populateSearchBar and parses the json and stores the workflow label and URI in an array (workflowSuggestions). Each entry in the array is a map in the following format: label:<workflowLabel>, uri:<workflowURI>. This list of suggestions is all that is needed to enable the autocomplete functionality in our react component. Once you’re done with your changes, make sure to rebuild the bundle.js file with the “./node_modules/.bin/webpack -d --watch” command. More documentation on the react component we used can be found at https://github.com/moroshko/react-autosuggest.

Variables

workflowSuggestions	Array containing the workflow labels and corresponding URIs. Each entry in the array is stored in the format: {label:<workflowLabel>, uri:<workflowURI>}

Functions	
escapeRegexCharacters	Input: string ‘str’ Output: regex escaped string Escapes user input to be treated as a literal string within a regular expression
getSuggestions	Input: user input in the search box ‘value’ Output: list of suggestions that match the user input Given a user input, find all the values in workflowSuggestions that match the user input and return them.
getSuggestionValue	Input: suggestion from the list of suggestions ‘suggestion’ Output: string Returns an input value when a suggestion is clicked. Right now, we are returning the label of the suggestion.
renderSuggestion	Input: selected suggestion ‘suggestion’, user input ‘query’ Output: string or react element Defines how suggestions are rendered. Currently, we encapsulate the matching portion of the suggestion with tags and return it as a react element.
parseAutocompleteData	Input: json file ‘res’ Output: none We parse res and insert an entry for every label+uri pair into our workflowSuggestions array.
componentDidMount	Input: none Output: none

	Invoked immediately after the component is mounted. Here, we initialize our workflowSuggestions.
onChange	Input: Action Event ‘event’, input ‘newValue’, string ‘method’ Output: none Changes the ‘value’ variable in our react component to match the user input.
onSuggestionsFetchRequested	Input: user input ‘value’ Output: none Called every time we need to update the suggestions (each time the user modifies the input).
onSuggestionsClearRequested	Input: none Output: none Clears the dropdown list of suggestions
onSuggestionSelected	Input: the selected suggestion ‘suggestion’, the value of the suggestion ‘suggestionValue’, index in the array ‘suggestionIndex’ Output: none Called when a suggestion is selected. We store workflowSuggestions and the selected suggestion’s URI and label in local storage, then change pages to workflow-main.html.

workflow-searchbar.js

This file contains the jQuery UI Autocomplete functions used in the search bar in the workflow-main.html page. The functionality is the same as the React component in the index.html page, but it uses jQuery instead.

Functions	
monkeyPatchAutocomplete	Input: none Output: none Overriding the _renderItem function in the default jQuery UI Autocomplete widget so that we can highlight the part of the suggestions that

	the user input matches

workflowInfoScript.js

This file contains all of the functions necessary for adding the panels of information to the page when a user clicks on one of the nodes in a visualization on a workflow page. This adds panels to the page in two separated sections, one for process information, and one section for variable information. These get added to the right side of the page and will dynamically resize the visualization graph as needed.

Variables	
processInfosCount	Counter for how many process information panels are currently showing on the screen (range 0-4)
processInfosIndex	Counter for how many process information sections have been added to the page. Counts up so that every panel added has a unique index.
variableInfosCount	Counter for how many variable information panels are currently showing on the screen (range 0-4)
variableInfosIndex	Counter for how many variable information sections have been added to the page. Counts up so that every panel added has a unique index.
variableSectionsShowing	List of the names of the variable information panels that are currently showing.
processSectionsShowing	List of the names of the process information panels that are currently showing.
$template	jQuery object for the template of each process information panel.
$templateVariables	jQuery object for the template of each variable information panel.
$vis	jQuery object for the visualization container to be used for resizing it dynamically.

Functions

addProcessInfo	Inputs: process URI string ‘processURI’, array of process input variables’ URIs ‘inputsArray’, array of process output variables’ URIs ‘outputsArray’ This checks for the number of already showing process information panels: - If 0 variable and process panels are showing, it resizes the visualization to make room for the information panels - If 4 are showing the top one (least recently added panel) is removed to make room for the new panel - Additionally, the process name is checked for duplicates so it does not get displayed twice A new panel is created by making a copy of the template and populating the information about the process’s name, and input and output variables list before it is added to the page.
addVariableInfo	Inputs: variable URI string ‘variableURI’, array of process URIs that use the variable ‘usedBy’, process URI of the process that generates this variable ‘generatedBy’, and ‘variableType’ This checks for the number of already showing variable information panels: - If 0 variable and process panels are showing, it resizes the visualization to make room for the information panels - If 4 are showing the top one (least recently added panel) is removed to make room for the new panel - Additionally, the variable name is checked for duplicates so it does not get displayed twice A new panel is created by making a copy of the template and populating the information about the variable’s name and information before it is added to the page.
clearAllPanels	Inputs: none Outputs: none Removes all of the information panels (variable and process) from the page, resets counters and arrays keeping track of displayed information sections.

removeVariableInfo	Input: name of the variable who's panel is to be removed 'removeID' Removes the panel associated with that variable ID from the page and from the arrays keeping track of their information.
removeProcessInfo	Input: name of the process who's panel is to be removed 'removeID' Removes the panel associated with that process ID from the page and from the arrays keeping track of their information.