

Workbench Dashboard – Managing Experiments using Data and Software Provenance

Tom Wong

A Report

submitted in partial fulfillment of the
requirements of the degree of

Master of Science in Computer Science & Software Engineering

University of Washington Bothell

2018

Project Committee:

Michael Stiber, Chair

Hazeline Asuncion

Johnny Lin

Table of Contents

1. Abstract.....	4
2. Introduction	5
3. Related Works.....	8
3.1. Provenance	8
3.2. Provenance Models	8
3.3. Serialization of Provenance Models	9
3.4. Data Provenance Visualization	10
3.4.1. Standalone Provenance Visualizer	11
3.4.2. Embedded Provenance Visualizer	13
3.5. Software Provenance Visualization	14
3.6. Problem Statement.....	15
3.7. Provenance Visualization in Workbench Dashboard	16
4. Methods.....	17
4.1. Architecture	17
4.2. Visualization.....	20
4.2.1. Design of Visualization.....	20
4.2.2. Implementation of Visualization	23
4.3. Design of the Interactions	24
4.3.1. Showing Node Labels or Edge Labels	25
4.3.2. Moving nodes	25
4.3.3. Moving and scaling the display window	26
4.3.4. Control Panel	26
4.3.5. Highlighting nodes related to an activity.....	27
4.3.6. Comparing two artifacts	28
4.4. Software Development Lifecycle	30
4.4.1. Usability Evaluation	30
5. Results.....	33
5.1. Identifying UI button and provenance information	33
5.2. Preference between interactive and non-interactive visualization	34
5.3. Positive Feedback	34
5.4. Negative Feedback.....	34
5.5. Potential Improvements.....	36

6. Discussion/Conclusion	37
6.1. Future Work.....	37
6.1.1. Improvements on the visualization	37
6.1.2. Improvements on the artifacts comparison feature	38
6.1.3. Applying Workbench Dashboard to different simulation software	38
7. References	39

1. Abstract

In e-Science, scientists use computer programs and data to run simulations. This process uses and generates many artifacts including program code, executable software, input and output files. The complexity of the relationships among artifacts grows with time and making it difficult to comprehend the relationships among artifacts. In computational neuroscience, due to the high complexity of software and the long runtimes of simulations, being able to understand the relationship among artifacts is very important for scientists to validate their results and other people's results. Although some existing systems can relieve the pain by visualizing data provenance and managing the workflow of the experiments, they do not show software provenance in the visualization and do not use the visualization to help analysis of results. This project aims at creating a software system, called "Workbench Dashboard", to visualize the artifacts and relationships among artifacts, based on data provenance and software provenance, to help scientists to understand the relationships among artifacts faster, and analyze the results of experiments quickly based on the visualization. The usability evaluation shows that the visualization and the features in the Dashboard could help users search artifacts and their relationships easily. The average time spent on searching an artifact is 68.5 seconds. The overall accuracy is 96.67%. The participants in the evaluation had both positive and negative opinions about the application. Some of them said the visualization is intuitive. Some of them think the node spacing and lengths of labels sometimes make it difficult for users to identify nodes. The major future work includes improving the usability and using Workbench Dashboard to visualize artifacts included in simulations created by different simulation software applications.

2. Introduction

In e-Science, scientists use computer programs in their experiments to process massive data sets and get results. The artifacts used in an experiment may include executable programs, versions of the executable programs, source code of the executable programs, scripts, input and output data, and activities of running scripts or programs. The complexity of the relationships among artifacts depends on the workflow of experiments. Moreover, as time goes by, the relationships among different artifacts become very complex. Identifying the artifacts in experiments and understanding their relationships could be challenging and time-consuming.

The complex workflow of computational neuroscience causes identifying artifacts and understanding relationships among artifacts to be more challenging. Figure 1 shows a simplified example of a computational neuroscience workflow. The workflow may involve many updates of software since simulators are complex software and running a simulation may take a long time. This means it is difficult to ensure that a simulator is bug-free. Even after the classical software development stage, in step 7, new bugs may be discovered. Results may trigger re-design of mathematical models. Performance issues may also be discovered. These issues require more development cycles to fix the software. Therefore, it will produce more experiments and the relationships among artifacts become even more complex.

1. Choose a biological model to investigate.
2. Identify the mathematical models of that system.
3. Develop software to simulate the mathematical models.
4. Do more development to deal with specialized hardware.
5. Comparing simulation results with neuroscience data to discover and fix bugs.
6. Design simulation experiments and setup parameter files.
7. Run simulations and collect data.
8. Publish results.
9. Results are archived in a re-usable format.

Figure 1. A simplified example of a computational neuroscience workflow

In addition, many scientists would like to make sure other people's experiments have correct results since they want to develop their experiments based on other people's results. Thus, they want to be able to assess the quality, reliability, trustworthiness and reproducibility of simulations done by other people. One way to do the assessment is by checking the artifacts and their interrelationships.

To identify artifacts and understand the relationships among artifacts easier, some scientists describe and record their experiment using scientific workflow management systems, such as Taverna [1], Kepler [2], AVOCADO [3] and VisTrails [4]. A scientific workflow is the recording of computational tasks and the dependencies among the tasks in an experiment. An example of a simple workflow is shown in Figure 2.

1. Select and retrieve data from a database.
2. Reformat the data.
3. Analyze the data and output the analysis results.

Figure 2. A simple workflow

There are three tasks in the example. Every task could be done by running built-in functions in workflow management systems, calling external scripts or invoking executable programs. In computational neuroscience, workflow management systems could record the tasks, the dependencies among tasks, the input and output data, the built-in functions in workflow management systems, external scripts and executable programs involved in a workflow.

However, workflow management systems do not show the software versions, the relationships among software versions and their relationships with other artifacts, which can provide in-depth information for scientists to understand and analyze the results of their experiments. For example, the different output results of two experiments with the same input data and different software versions could mean potential defects in the newer version of software.

The purpose of this project is to employ interactive provenance visualization to help scientists identify and understand the relationships among artifacts easily, and potentially help the analysis of their results. This project utilizes both software and data provenance to visualize the

relationships among artifacts used in neural network simulations run by BrainGrid+Workbench [5]– a toolkit/software architecture to create high-performance neural network simulator. The purpose of the visualization is to assist scientists on checking information about simulation artifacts and relationships among artifacts, like software versions, dependencies among software versions, input files used by simulations, output files generated by simulations, number of simulations, and the start time and end time of the simulations.

Workbench Dashboard is part of a research project, called “BrainGrid+Workbench” [5], under the direction of Dr. Michael Stiber at the University of Washington Bothell. BrainGrid is a high-performance neural simulator. Workbench has a user-friendly graphical user interface (GUI) for configuration and management of data and parameters for high-performance neural simulations. Besides the GUI, it enables queries for data provenance. This project is to visualize data provenance and software provenance to help BrainGrid Workbench users manage simulation artifacts and analyze simulation results.

3. Related Works

3.1. Provenance

According to Merriam-Webster Dictionary, provenance is “the history of ownership of a valued object or work of art or literature” [6]. However, it is also used in computer science to record the history of data and software. According to the World Wide Web Consortium (W3C), provenance is “information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness” [1].

The provenance in experiments includes data provenance and software provenance. Data provenance includes data artifacts, such as input and output data, executable programs, scripts and activities of running scripts or programs, and relationships among the data artifacts. Software provenance includes software artifacts, like versions of the source code, source code of the executable programs and the dependencies among software artifacts.

This project visualizes both data and software provenance to help scientists understand the artifacts and their relationships in simulation experiments created by BrainGrid+Workbench [5].

3.2. Provenance Models

To help ease storage and sharing of provenance data, two provenance models – open provenance model (OPM) [7] and W3C provenance data model (PROV-DM) [8]– have been used as standards to store provenance.

Figure 3 shows the basic elements in PROV-DM – agent, entity, and activity – and the relationships among them.

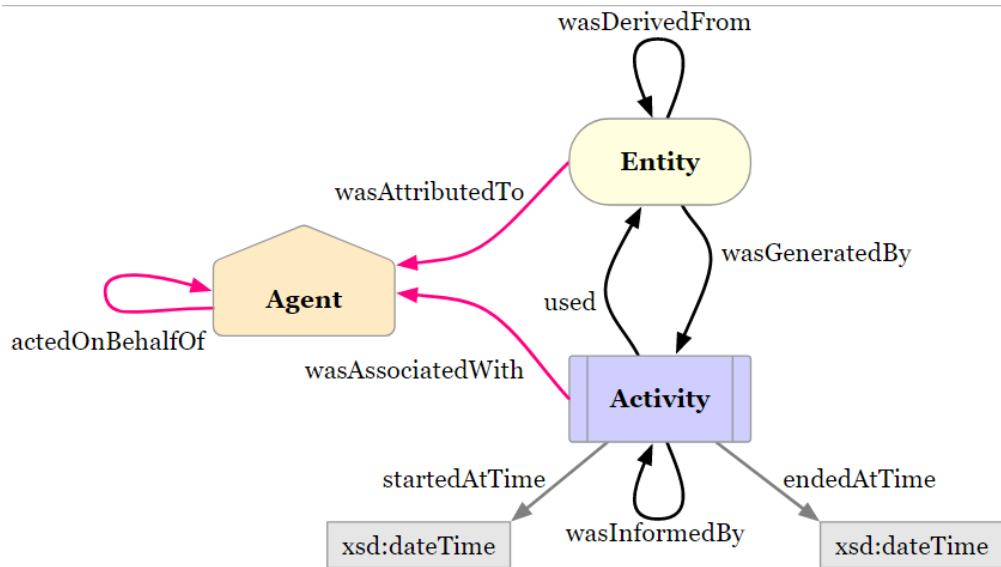


Figure 3. The basic elements of PROV-DM [8] and the visualization in PROV-O [9]

According to W3C [8], an agent is anything that is responsible for doing an activity, for the existence of an entity, or for another agent's activity. An activity has a start time and end time. It takes input entities and produces output entities during the activity. An entity is a physical, digital, conceptual, or other kind of thing with some fixed aspects. A simple example is running a program to reformat a data file. The program is the agent, which is responsible for the reformatting activity. The reformatting activity takes an unformatted file as an input entity and produces a formatted file as an output entity.

3.3. Serialization of Provenance Models

To store provenance using PROV-DM, W3C created the PROV Ontology (PROV-O) [9] standard to map PROV-DM to Resources Description Framework (RDF) graph [10]. RDF graph uses subject-predicate-object triples to represent relationships between artifacts. Referring to Figure 3, an activity, agent or entity can be the subject or object in a triple. A relationship among activities, agents or entities can be a predicate. For example, "Activity 1 used Entity 2" is a subject-predicate-object triples. In the example, "Activity 1" is the subject. "used" is the predicate. "Entity 2" is the object. RDF graph can be stored in different formats, like Turtle [11], XML and N-Triples.

In this project, Workbench Dashboard reads provenance from a RDF file in Turtle format with the PROV-O standard to retrieve provenance information.

3.4. Data Provenance Visualization

Data provenance visualization visualizes the data artifacts and the relationships among data artifacts. There are two types of provenance visualizers – standalone provenance visualizer and embedded provenance visualizer. Table 1 summarizes the features of different data provenance visualization tools and Workbench Dashboard.

Table 1. Features of different data provenance visualization tools and Workbench Dashboard

	W3C RDF Validation Service [12]	PROV-O-Viz [13]	Komadu [14]	AVOCADO [3]	VisTrails [4]	Workbench Dashboard
Node-link diagram	X		X	X	X	X
Interactive visualization		X	X	X	X	X
Support PROV-O standard	X	X	X			X
Visualizing activities	X	X	X	X	X	X
Visualizing entities	X	X	X			X
Visualizing agents	X	X	X			X
Visualizing software versions						X
Comparing artifacts						X
Highlighting activity cluster						X

3.4.1. Standalone Provenance Visualizer

Standalone provenance visualizers can visualize provenance stored with the provenance data model standards mentioned in Section 3.3. W3C RDF Validation Service [12], PROV-O-Viz [13] and Komadu [14] can visualize the provenance stored in an RDF file with the PROV-O standard. As shown in Figure 4, W3C RDF Validation Service visualizes a provenance graph as a static node-link diagram containing all the artifacts and their relationships in a single image. The nodes and links in the diagram represents the artifacts and the relationships among them. However, since it is a static image, all provenance information is put into a single image. Therefore, the image will be very large if the size of provenance information becomes larger and may cause inconvenience for users to search information in the image since they may need to drag the image and zoom in and out more frequently to search for information.

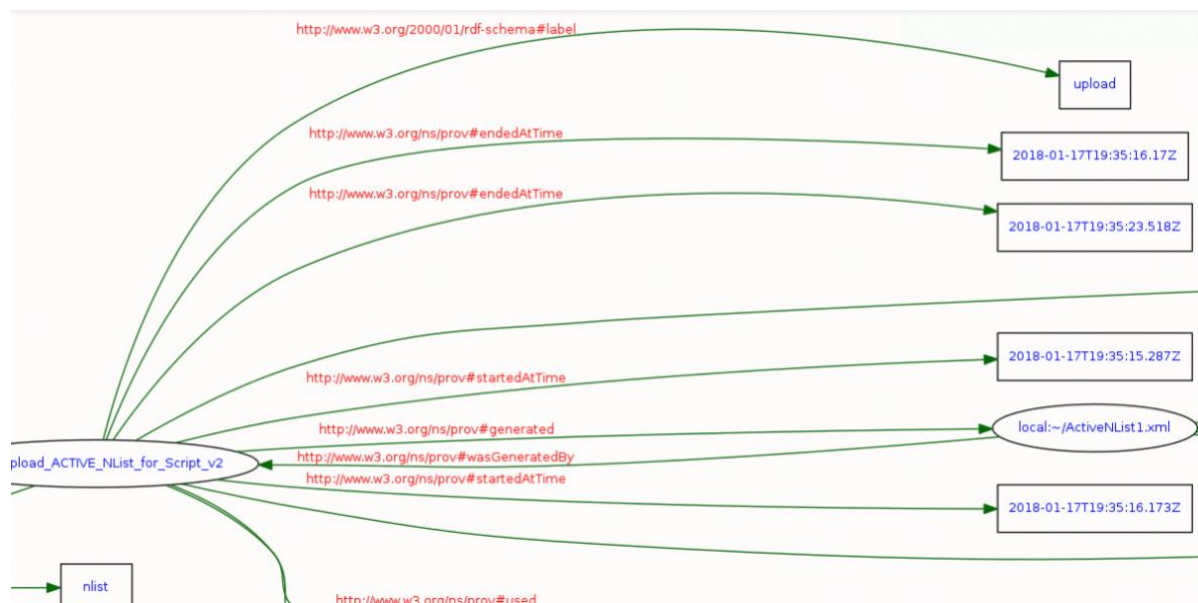


Figure 4. Part of a node-link diagram generated by W3C RDF Validation Service [12]

Figure 5 shows the interactive Sankey Diagram created by Prov-O-Viz to visualize provenance. It shows the data flows among entities. It also displays different kinds of additional information when users move their cursor over the graph. Users can also drag the bars to positions they like.

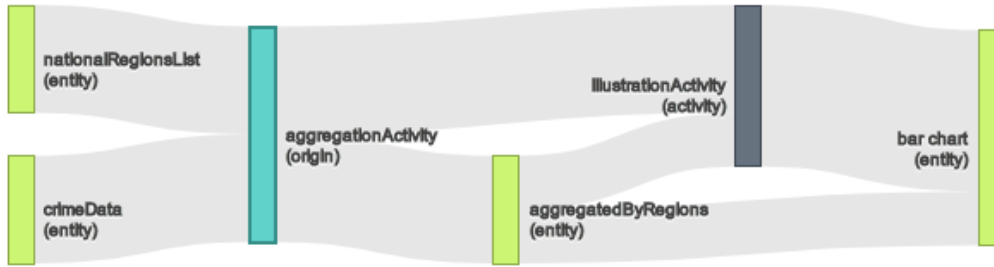


Figure 5. A Sankey Diagram created by PROV-O-Viz [13]

Figure 6 shows the provenance graph generated by Komadu [14] and visualized by a visualization tool called “Cytoscape” [15]. Besides visualizing provenance stored in files, Komadu can work with different systems to collect provenance and generate provenance graph. It has an Ingest API exposed as a Web Service so that external systems can pass data provenance to Komadu when it calls the API. Then, Komadu can generate provenance graph and store in a CSV file, which can be visualized by Cytoscape.

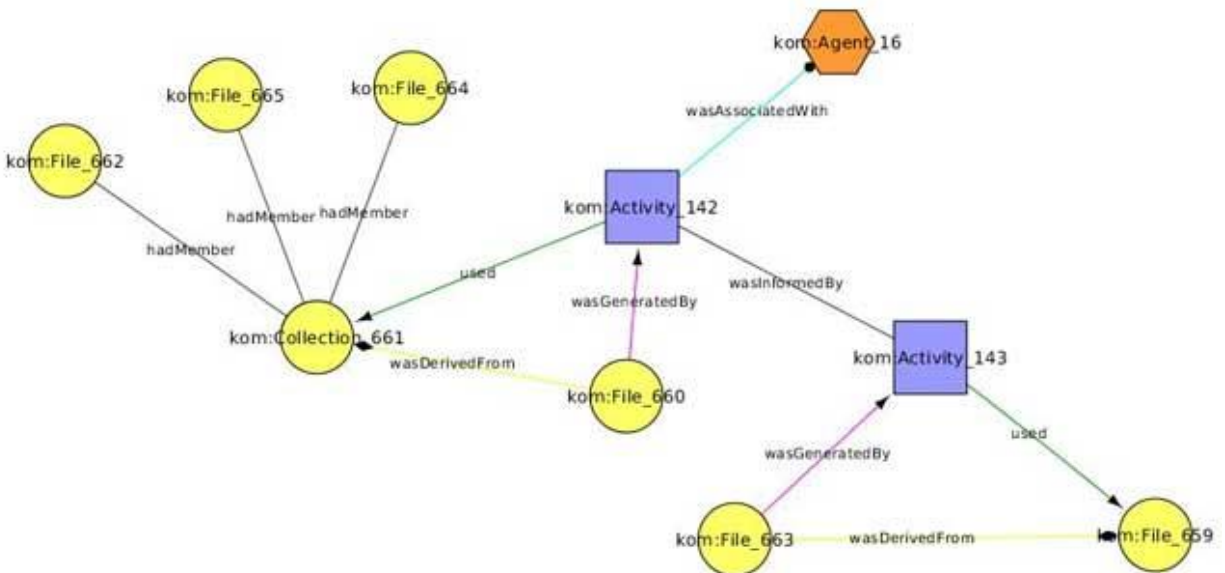


Figure 6. A provenance graph created by Komadu [14]

The advantage of standalone visualizer is the high portability. It does not need to generate provenance information. It can visualize any provenance file as long as the provenance file meets the compatible provenance standards. However, the disadvantage is that it cannot

control the provenance information captured in a provenance file. It cannot visualize the missing information in a provenance file.

3.4.2. Embedded Provenance Visualizer

Many scientific workflow management systems can visualize workflow and data provenance besides managing and running scientific workflows. AVOCADO [3] and VisTrails [4] can capture data provenance when running scientific workflows and then create interactive visualization of the data provenance. Figure 7 shows the interactive visualization of a provenance graph in AVOCADO [3].

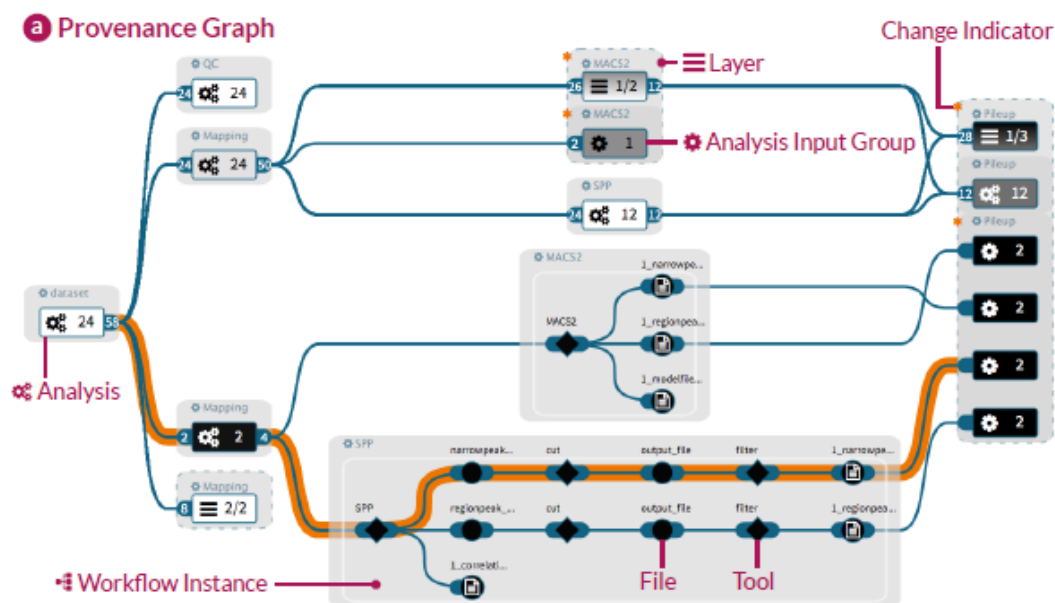


Figure 7. A provenance graph in AVOCADO [3]

By interacting with the system, users can view information from high level to low level. By default, similar nodes are merged into one node to allow users to understand the overall provenance. Then, the users can view the detailed nodes by clicking on a merged node. This visualization is useful when the provenance graphs is too large to fit into users' screens. In addition, users can focus on nodes they are interested in and would not be distracted by information they are not interested in.

Figure 8 shows the interactive visualization of a provenance graph in VisTrails [4]. The provenance graph is based on a workflow created by a user. It shows how input data flow from

the fetch input activities – “DownloadFile”, “vtkLookupTable”, “vtkCamera” and “vtkProperty” to the final output activity “vtkRendererOutput”.

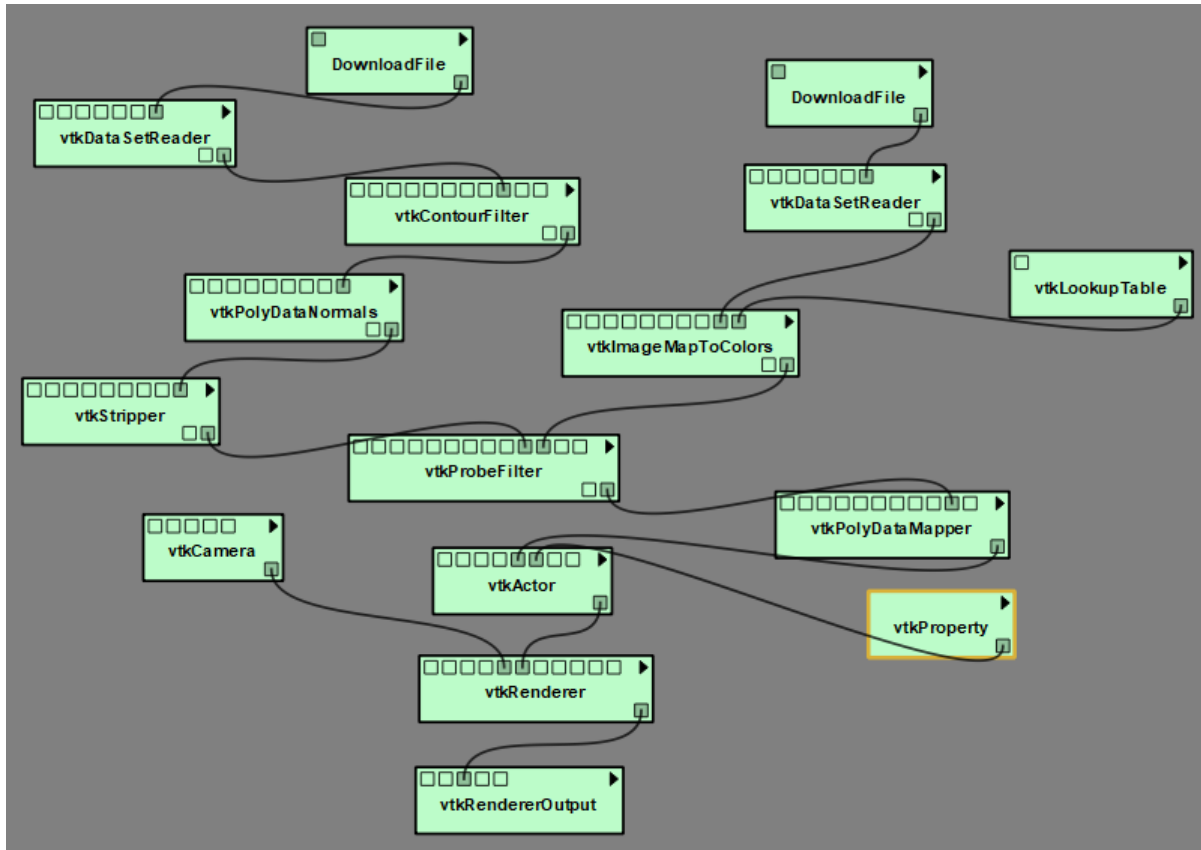


Figure 8. A data provenance graph in VisTrails [4]

The advantage of embedded provenance visualizer is the granularity of provenance information. It can control every piece of provenance information displayed in its visualizations. However, the disadvantage is the visualization is not portable. It cannot visualize provenance file generated by other systems.

This project used the approach of standalone provenance visualizer since the visualization can be applied to provenance files generated by different systems in the future. Workbench Dashboard visualizes data provenance stored in a RDF Turtle file with PROV-O standard.

3.5. Software Provenance Visualization

Software provenance visualizer visualize dependencies among software versions in a commit graph. Figure 9 shows a commit graph created by GitHub [16].

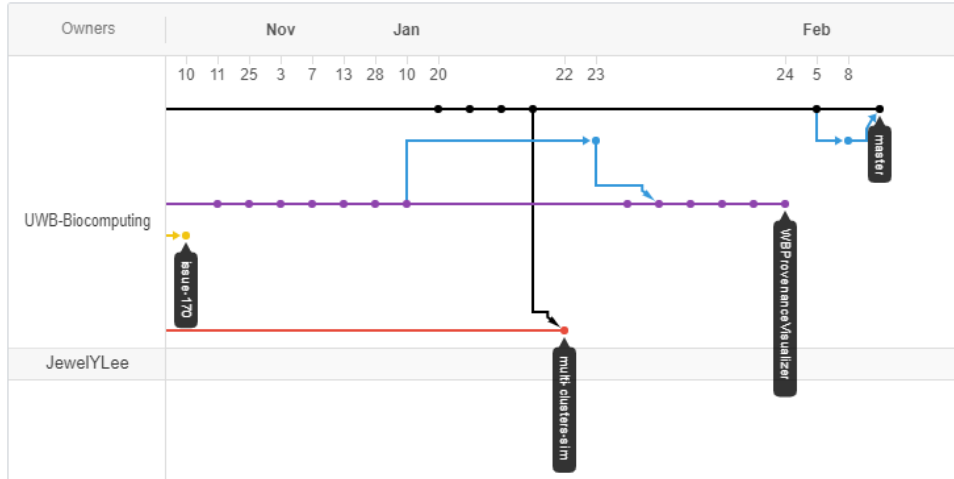


Figure 9. A software commit graph created by GitHub [16]

The commit graph shows the dependencies among software versions, also known as commits. Every dot represents a commit. The lines connecting dots represent the dependencies among commits. Different colors represent different branches in the repository. The time line at the top shows the commit time of each commit. This visualization is similar to a node-link diagram, which can visualize the dependencies among nodes. The dots are the nodes. The lines are the edges.

This project used node-link diagrams to visualize software provenance.

3.6. Problem Statement

The above visualization tools can visualize either data provenance or software provenance. The problem of only visualizing data provenance is that users cannot utilize the dependencies among software versions. Figure 10 shows an example of this situation. In the example, commit 6 fixed a bug, which exists in the system since commit 4. If users know the dependencies among commit 4 to commit 6, they may want to check if the bug has any impact to the old results by re-running experiment 1 and 2 using commit 6. However, if they cannot get the dependencies among the commits from the software provenance visualization, they do not know which experiments may be potentially affected by the bug.

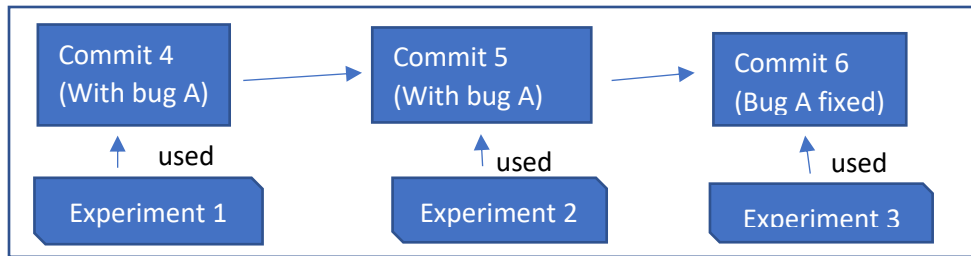


Figure 10. An example of dependencies among commits used in experiments

On the other hand, the problem of only visualizing software provenance is that users cannot know the input files, output files and commits used in experiments.

3.7. Provenance Visualization in Workbench Dashboard

This project addresses the problem stated in Section 2.6 by visualizing both data provenance, software provenance and the connection between them. The visualization can show dependencies among software versions, which are used to create executable files. The executable files take input data or generate output data during experiments. Workbench Dashboard also employs some techniques in the systems mentioned in Section 2.4. Like standalone visualizers in Section 2.4.1, it reads provenance file with W3C PROV-O standard [9]. Moreover, it visualizes provenance as an interactive node-link diagram as mentioned in Section 2.4.2. In addition, it has additional features to help users comprehend the relationships between artifacts used in neural simulations created by BrainGrid+Workbench [5] and analyze the simulation results. A highlighting feature can emphasize the most relevant nodes of an activity. An artifacts comparison feature can compare pairs of text files artifacts in a side-by-side text view.

4. Methods

Workbench Dashboard is written in Java 8. The user interface is built by JavaFX [17]. The project uses the following open source libraries: JGit [18], RichTextFX [19], ControlsFX [20] and DiffUtils [21]. JGit is used to extract dependencies between commits from the Git repository of BrainGrid+Workbench. RichTextFX is used to highlight texts with different colors in the side-by-side text view to compare two artifacts. ControlsFX is used to create some of the user interface elements, like toggle buttons and sliders. DiffUtils is used to compute the differences between two artifacts. Using these libraries sped up the development process and increases the modularity of the system since different libraries have their own responsibilities and can be replaced by similar libraries without changing many lines of code.

4.1. Architecture

Figure 11 shows an overview of the Workbench Dashboard architecture. BrainGrid+Workbench contains BrainGrid and Workbench. BrainGrid is a simulator responsible for running simulations. Workbench is a management tool to create input files and invoke the simulator to run simulations remotely or locally. It also responsible for collecting simulation results and provenance information. Then, it stores the provenance information into RDF Turtle files. The local Git repository contains software provenance for the simulator, such as source code, dependencies between software versions (also called “commits” in Git) and commit messages. The Git repository is downloaded from the remote GitHub [16] repository. Workbench Dashboard could retrieve the software provenance from the local Git repository without accessing the remote repository, which makes the retrieval faster. Workbench records the data provenance and the commits used to produce executable files to run simulations in a provenance RDF Turtle file with the W3C PROV-O standard. Workbench Dashboard uses the software provenance in the local Git repository and the provenance file generated by Workbench to instantiate objects for drawing a provenance graph. After that, it visualizes all the artifacts and their relationships.

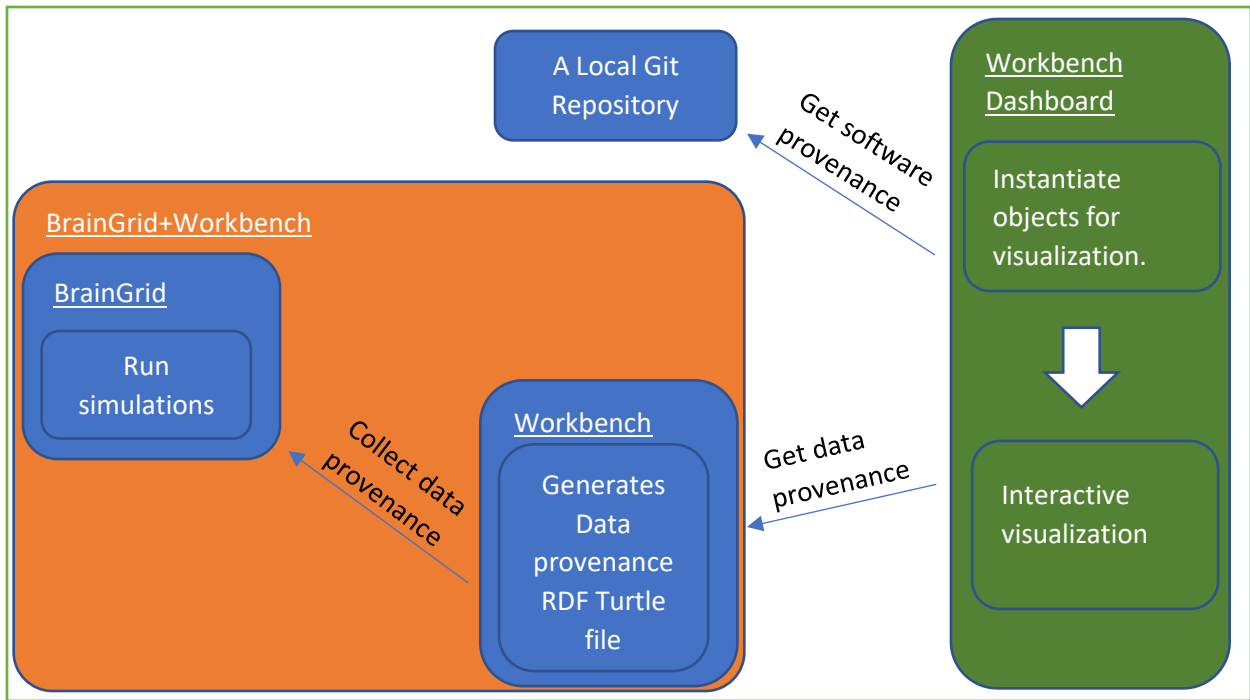


Figure 11 – An overview of the architecture of Workbench Dashboard

Figure 12 shows the data flow diagram of BrainGrid Workbench. The software provenance is extracted from the local Git repository by Workbench Dashboard. The data provenance is extracted from BrainGrid and stored in a RDF Turtle file. Workbench Dashboard extracts data provenance from the RDF Turtle file.

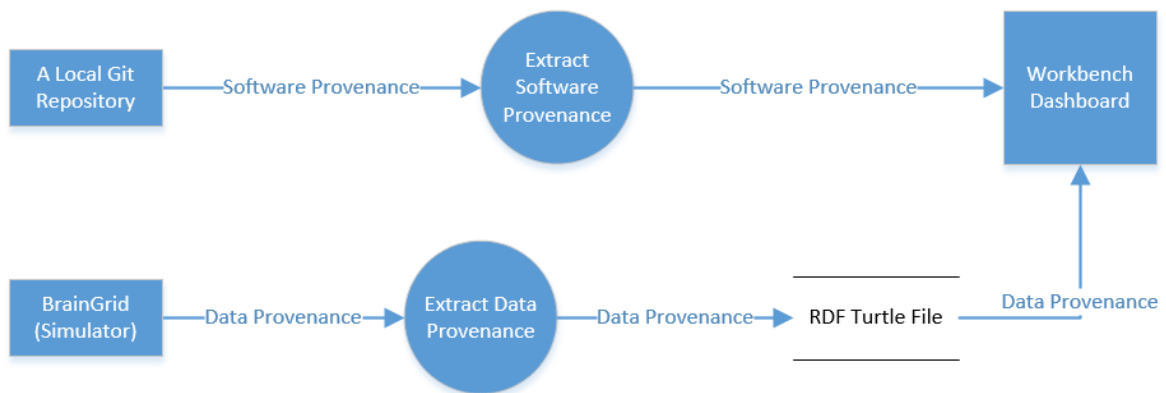


Figure 12. Data flow diagram of BrainGrid Workbench

Figure 13 shows a simplified UML class diagram of Workbench Dashboard, which uses model-view-controller design pattern.

In the Model classes, the Graph class consists of a list of Node and a list of Edge objects. The Node class consists of node ID, dimension, position and color of the node in the graph.

ActivityNode, AgentNode, CommitNode and Entity Node inherit the Node class. The Edge class consists of the node ID of the from-node and the to-node, and the relationships between from-node and to-node.

The ProvenanceVisualizerController class contains a Graph object, a VisCanvas object. The VisCanvas object is the view. The controller draws the visualization on the VisCanvas based on the data in the Graph object and the configuration of the visualization, like node spacing.

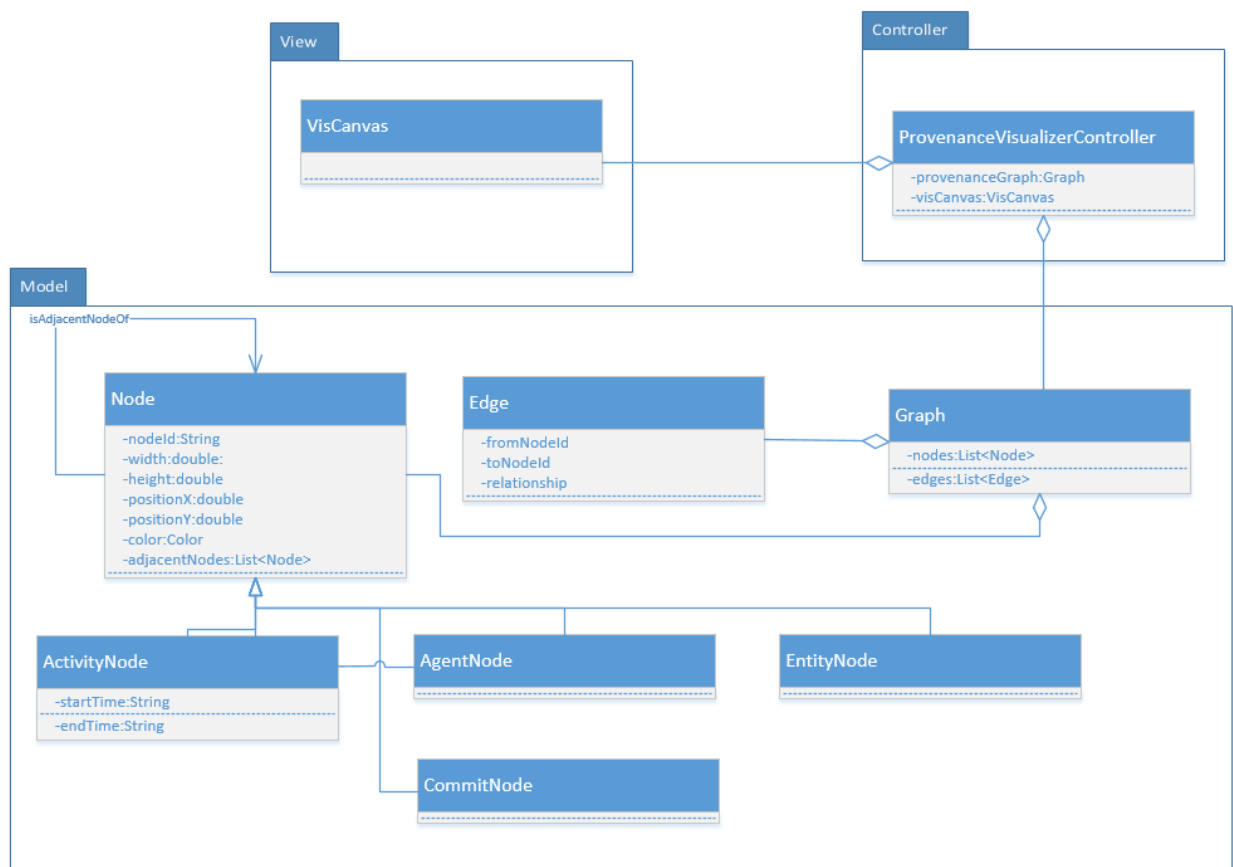


Figure 13. UML Class Diagram of Workbench Dashboard

4.2. Visualization

4.2.1. Design of Visualization

Workbench Dashboard visualizes data provenance graph stored in the RDF file as a node-link diagram. The reason of using node-link diagram is that node-link diagram can visualize relationships among nodes. Nodes represent artifacts, which are activities, agents and entities in the PROV-O standard. Edges represent relationships among artifacts. Figure 14 shows the initial visualization in the dashboard with nodes connected by edges. Nodes have different shapes and colors to represent different kinds of artifacts in the simulations. The shapes of nodes follow the W3C PROV-O standard, as shown in Figure 3, to distinguish activities, software agents and entities. In addition, different colors are used to identify different kinds of entity nodes – commit, input and output, as shown in Figure 15. The reason for using different colors and shapes is to help users identify different kinds of nodes easier [22].

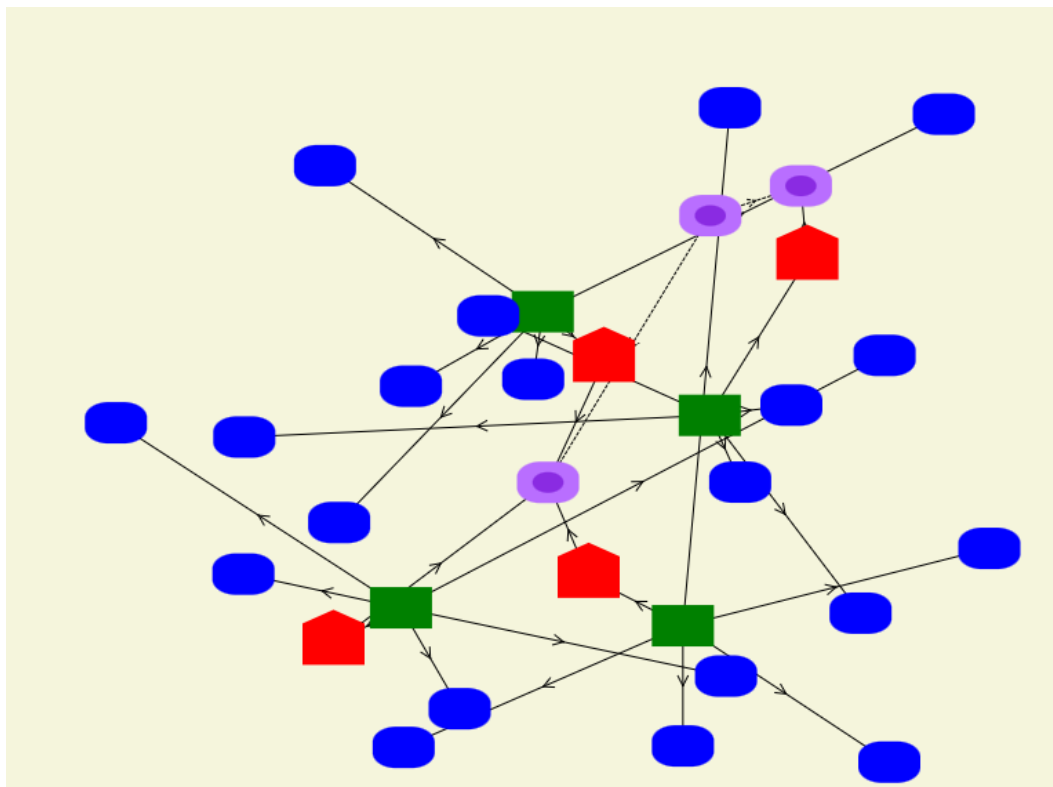


Figure 14. Initial visualization in the Workbench Dashboard

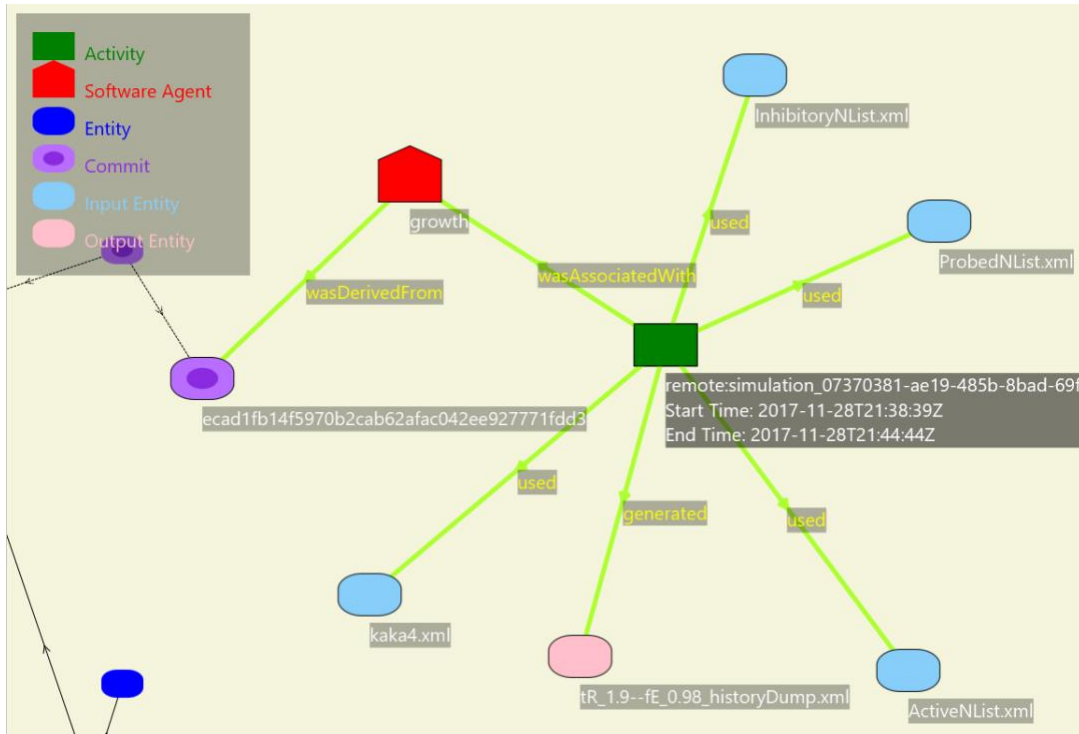








Figure 15. Visualization in the Workbench Dashboard

Every node has its own label to show its identity and other significant information. Depending on the kind of nodes, the values of labels have different meanings to provide users important information of nodes, as shown in Table 2.

Table 2. Meaning of label values of different kinds of nodes

Node Type	Meaning of label values
Activity 	<ol style="list-style-type: none"> 1. The name of the simulation 2. The start time of the simulation 3. The end time of the simulation
Software Agent 	The name of the executable file to run the simulation
Entity 	The file name
Input Entity	The input file name

	
Output Entity 	The output file name
Commit 	The commit ID of the commit in Git repository

The edges are directed edges with arrows to show the relationships among artifacts, like “used”, “wasAssociatedWith”, “wasDerivedFrom” and “generated”. For edges among commits, if one commit is the immediate parent of another commit, the edge between the two commits is a solid line. Otherwise, the edge is a dashed line. It can help users identify immediate parents faster. As shown in Figure 16, the commit starting with “32a89” is the immediate parent of the commit starting with “1666e”, and therefore the edge between them is a solid line. On the other hand, the commit starting with “eff23” is not the immediate parent of the commit starting with “32a89”, which means there are some commits between them. Thus, the edge between them is a dashed line to indicate that there are commits not being displayed in between.

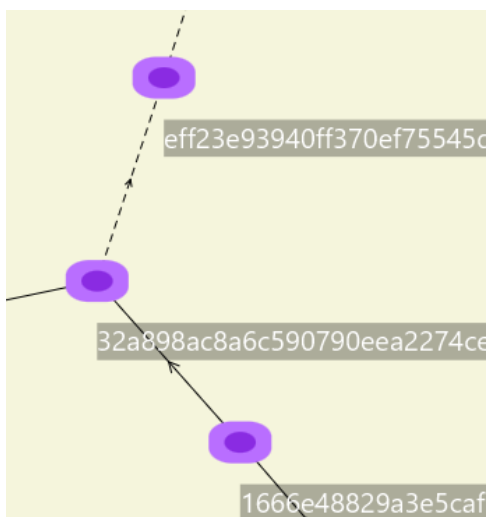


Figure 16. Edges among commits

The default color of edges is black. The green edges in Figure 15 are due to the highlighting effect in Section 4.3.5.

4.2.2. Implementation of Visualization

Before drawing the node-link diagram, Workbench Dashboard instantiates the underlying objects to store the nodes and edges. It reads the subject-predicate-object triple statements in the provenance RDF Turtle file generated by Workbench to create nodes for the subjects and objects. The subjects and objects are artifacts, like activities, agents, entities and commits. It also creates edges for the predicates. Edges are the relationships, like “used”, “generated”, “wasAssociatedWith” and “wasDerivedFrom”.

After that, it creates the edges among commit nodes by extracting the dependencies between commits from the Git repository using the JGit library [18]. Figure 17 shows the algorithm for creating edges among commit nodes. Initially, the variable “commits” contains an array of commits in the provenance file. The algorithm finds the missing common ancestors and inserts them into the commits array to make sure all commits can be connected and shown in a single graph. Then, the commits are sorted with descending commit times. After that, the algorithm iterates through the commits to create edges among commits.

```

commits = an array of commits in the provenance file;
//get a list of commits for common ancestors in commits using JGit
commonAncs = getCommonAncestors(commits);

foreach commit in commonAncs{
//add common ancestor to commits if it is not in commits
    if(!commits.contains(commit))
        commits.add(commit);
}

Sort commits by descending commit times
//branches is used to store the latest commit of each branch
branches = empty list;
foreach commit in commits {
    //removalList stores commits, which will be removed from branches later
    removalList = empty list;
    foreach branchCommit in branches {
        if (commit is the ancestor of branchCommit) {
            removalList.add(branchCommit);
            if (commit is the immediate parent of branchCommit) {
                create a solid edge between commit and branchCommit
            } else {
                create a dashed edge between commit and branchCommit
            }
        }
    }
}
//remove the commits in removalList from branches
branches.removeAll(removalList);
//add commit, which is the latest commit, to branches
branches.addLast(commit);
}

```

Figure 17. Pseudocode to create edges among commits

Finally, the nodes are stored in a hash maps with node IDs as the keys. Node IDs are the subject or object strings in the RDF Turtle file. The edges are stored in a hash map with edge IDs as the key. Edge IDs are the concatenation of subject, predicate and object strings in the RDF Turtle file. Hash maps allow fast access with $O(1)$ time complexity.

After instantiating the objects, the system draws the node-link diagram on a Canvas GUI component of JavaFX.

4.3. Design of the Interactions

Initially, the visualization in Workbench Dashboard looks like Figure 14, in which nodes are placed randomly. But, eventually, the nodes move to positions which minimize the intersection of edges, as shown in Figure 18. The final layout is less messy and easier for users to look for connections among nodes. The positions of nodes are calculated by a force-directed graph

layout algorithm [23], which calculates the attractive and repulsive forces on every node and move each node according to its net force.

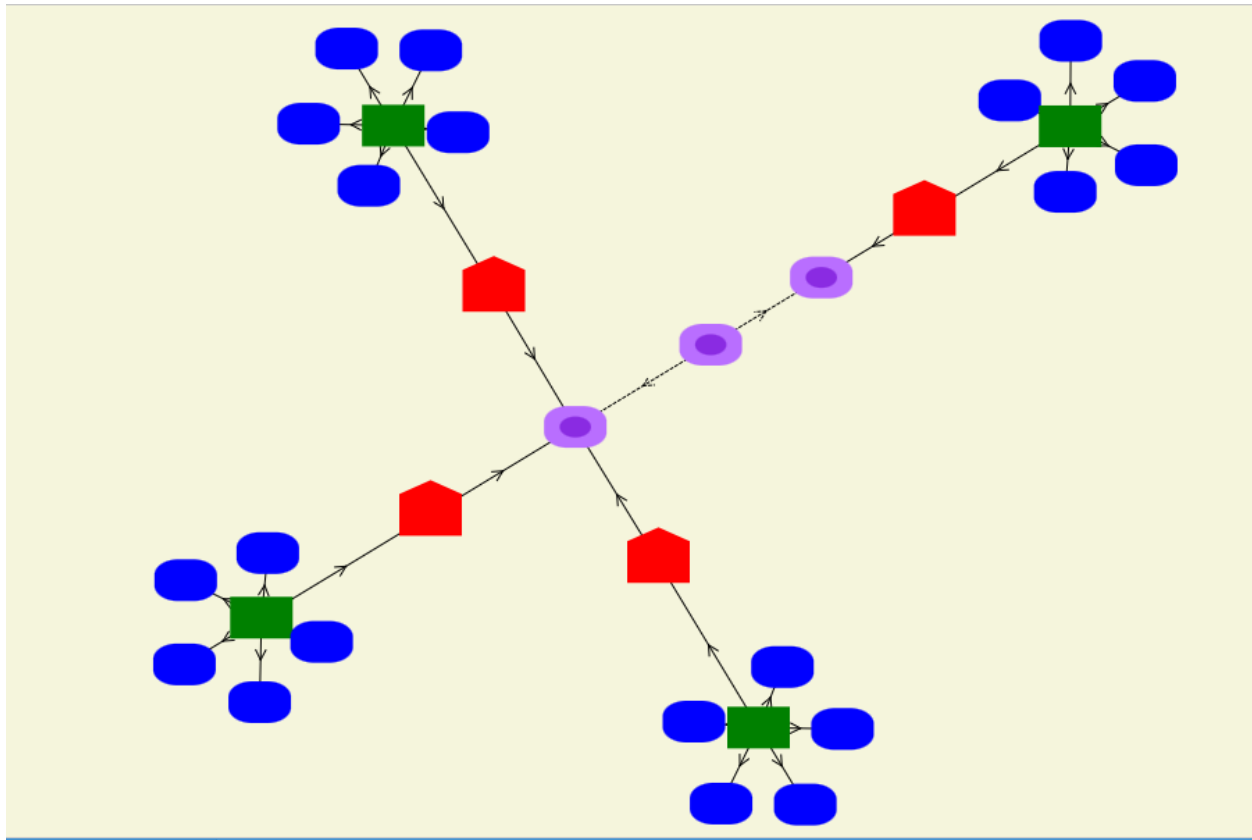


Figure 18. Final state of the graph

4.3.1. Showing Node Labels or Edge Labels

Users can check a label of a node or an edge by moving a cursor over the node or the edge. Labels disappear if the cursor moves away. If users click on a node or edge, the label is displayed even if the cursor moves away. This allows users to check labels fast and then choose to focus on particular pieces of information. The program uses the coordinates of a cursor and a node, and the size of the node to determine if the cursor is on a node.

4.3.2. Moving nodes

Users can move a node by dragging it to anywhere inside the display window. This allows users to choose which nodes they want to focus on. For example, users can move nodes to the edge

of the display window if they are not interested in the nodes. On the other hand, they can move nodes back to the center when they are interested in the nodes.

4.3.3. Moving and scaling the display window

Users can move the display window by dragging the space with no nodes and edges. They can zoom in or out to a specific part of the graph by using the scroll wheel. It allows users to navigate among different node clusters to check information.

4.3.4. Control Panel

The control panel is located at the right side of the display window. Figure 19 shows how it looks like.

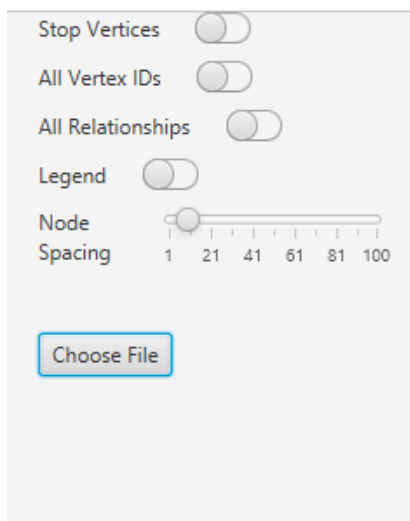


Figure 19. The control panel in the dashboard

The control panel enables users to change the following parameters:

1. Stop Vertices: stop the movement of nodes by stop applying the force-directed graph layout algorithm [23].
2. All Vertex IDs: show all node IDs, i.e., the node labels.
3. All Relationships: show all relationships between nodes, i.e., the display names of the edges.
4. Legend: show the legend at the top left corner, which explains the meaning of each type of nodes, as shown in Figure 15.

5. Node Spacing: control the space between nodes, i.e., the repulsive force between nodes. Greater spacing implies greater repulsive force.
6. Choose File: choose a data provenance file in RDF Turtle format to be loaded into Workbench Dashboard.

4.3.5. Highlighting nodes related to an activity

If a user places the cursor on an activity node, nodes and edges related to the activity are highlighted. Figure 20 and Figure 21 shows the appearance of the nodes and edges before and after applying the highlighting effect. After applying the effect, all the related entity nodes, agent nodes and activity nodes change to larger sizes. The input entity nodes change to blue. The output entity node changes to pink. The labels of the nodes also appear to let users identify the important artifacts related to that activity. The edges are thicker and change to green color. If users click on the activity node, the highlighting effect remains on the screen even after moving the cursor outside the node. This function allows users to check activity node and its related nodes quickly and choose to keep the effect with one click. Users do not need to select all the related nodes manually by clicking on each individual node. It can save users time on clicking and searching for related nodes.

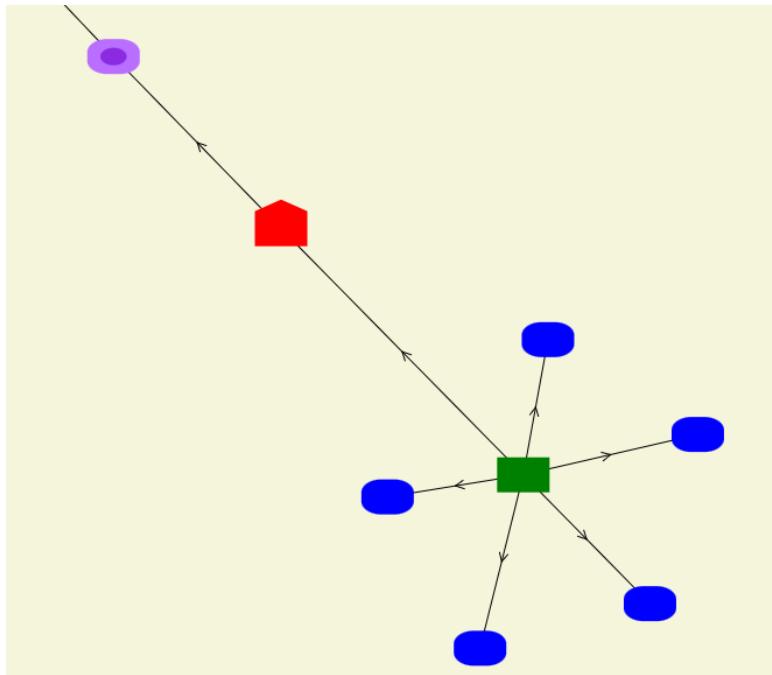


Figure 20. Before highlighting nodes related to an activity

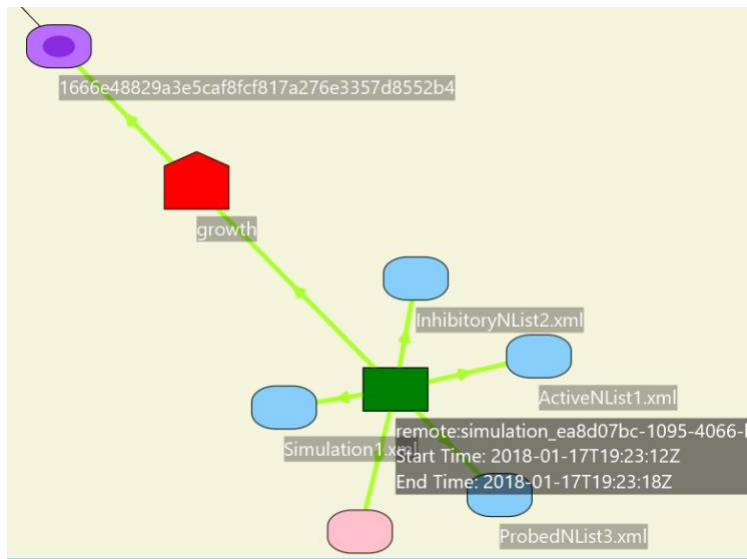


Figure 21. After highlighting nodes related to an activity

The following steps are used to search for nodes related to an activity.

1. Input entities: Search for the edges with key strings starting with the activity node ID + “used”. The “used” relationship indicates the to-node of the matching edges are input entities.
2. Output entities: Search for the edges with key strings starting with the activity node ID + “generated”. The “generated” relationship indicates the to-node of the edges are output entities.
3. Software Agents: Search for the edges with key strings starting with the activity node ID + “wasAssociatedWith”. The “wasAssociatedWith” relationship indicates the to-node of the matching edges are software agents.
 - 3.1. Commits: Search for the edges with key strings starting with the agent node ID + “wasDerivedFrom”. The “wasAssociatedWith” relationship indicates the to-node of the matching edges are the commits.

4.3.6. Comparing two artifacts

Users can compare two artifacts by dragging one node to another node, as shown in Figure 22. The color of the comparing node (i.e., Result3.xml) is changed to yellow if it is close to the

dragged node (i.e., Result2.xml). The changing of color is designed to help users distinguish the comparing node.

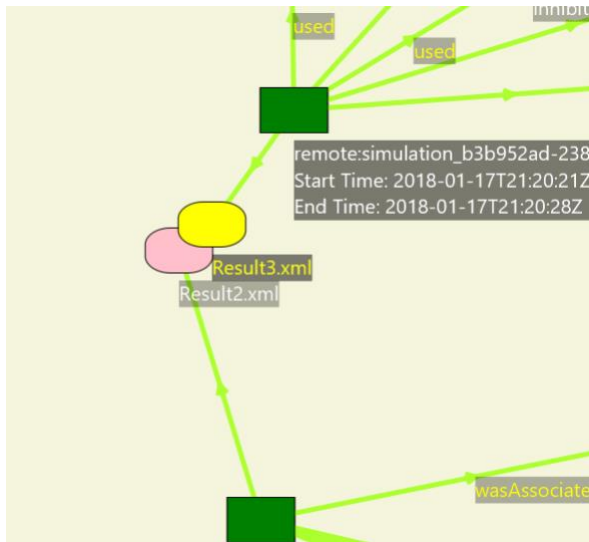


Figure 22. Comparing two artifacts by dragging one node to the other

After users release the node, Workbench Dashboard checks if the two files exist in the local file system. If either file does not exist, the dashboard downloads the missing files via SFTP protocol. (The user may need to input his credentials to access remote files.) After downloading the missing files to the local file system, the system uses the Myer's diff algorithm implemented in the DiffUtils library [21] to calculate the different lines between the two files. The different lines are displayed in a side-by-side text view and color-coded to represent the inserted, changed and deleted lines, as shown in Figure 23. Red lines represent inserted lines. Green lines represent changed lines. Gray lines represent deleted lines. The side-by-side text view and color-coded lines can help users identify differences between two text files at a glance.

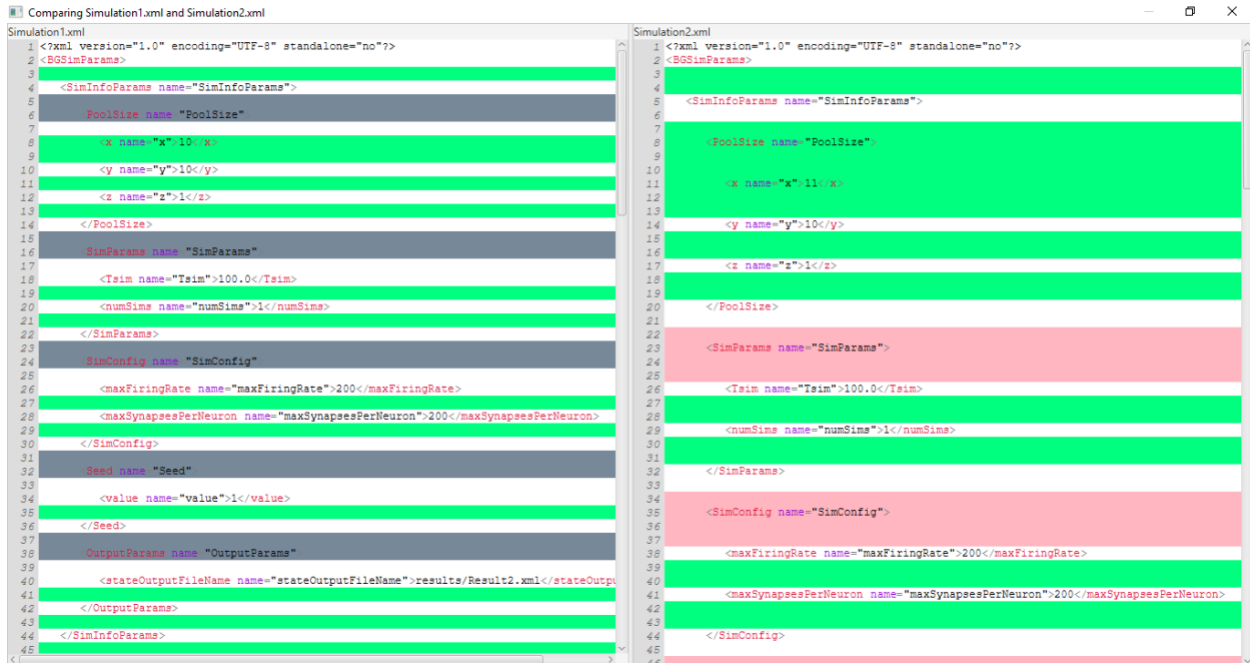


Figure 23. A Side-by-side text view to compare two artifacts

4.4. Software Development Lifecycle

The software development lifecycle was an Agile development lifecycle, which produced executable software continuously in each cycle. The length of a sprint was around one to two weeks. A retrospective was done at the end of each sprint to review the development process to improve efficiency of the development and quality of the software. After that, new features of visualization were determined, prioritized and added to the product backlog.

4.4.1. Usability Evaluation

A usability evaluation was conducted to collect data about how users interact with Workbench Dashboard, what is beneficial to users and what could be improved. Three individuals with significant neuroscience expertise participated in the evaluation. They were given a 10-minute brief introduction of Workbench Dashboard and then were asked to identify the artifacts, simulations and their relationships in the Workbench Dashboard. Participants' feedback and their interactions with the dashboard were recorded.

Table 3 shows the questions asked in the usability evaluation. Q1 asked participants to identify the toggle button to turn on the legend. Q2 to Q10 asked the participants to identify various

provenance information in the visualization created by Workbench Dashboard. The provenance information was based on three simulation experiments run by BrainGrid, using different parameters in the input files. The simulations used models described in [24]. One simulation used 110 neurons. Two simulations used 100 neurons. These simulations were performed only to generate artifacts and provenance for the usability study. Q11 asked if the participants prefer to use the non-interactive graph generated by the W3C RDF Validation Service [12] or the visualization in Workbench Dashboard to search for information. Q12 and Q13 asked for the positive and negative feedback about Workbench Dashboard. Q14 asked the potential improvements for Workbench Dashboard.

Table 3. Questions asked in the usability evaluation

Q1	Turn on the Legend using the toggle button at the right side.
Q2	What is the activity with start time 2018-01-17T19:23:12Z?
Q3	What are the commits, input entities, output entities and software agents used in the activity mentioned in Q2?
Q4	What is the activity, which used the input file "Simulation2.xml"?
Q5	What are the output entities used in the activity mentioned in Q4?
Q6	Drag the node "Result1.xml" to "Result2.xml" to compare the output entities in the two activities. Do you find any differences?
Q7	For the above mentioned two activities, which activity used a more up-to-date commit?
Q8	What is the name and end time of the activity, which used "Simulation3.xml"?
Q9	Drag the node "ActiveNList3.xml" to "ActiveNList1.xml" to see their differences. Do you find any differences?
Q10	What is the common ancestor of the commits used in remote:simulation_b3b95... and remote:simulation_ff632...?
Q11	Comparing with the image in the following link, do you prefer searching for information in the Workbench Dashboard?

	https://drive.google.com/file/d/1kkVS4gBH2nLDKZ7T3Cggsr7HHvmwzt2t/view?usp=sharing
Q12	What do you like about the application?
Q13	What do you dislike about the application?
Q14	Do you find any potential improvements for the application?

The metrics used to evaluate the usability were as follows:

- Correctness of the participants' answers.
- Time for the participants to identify different artifacts, like the simulation activities, input files, output files and commits, and their relationships in the simulations.
- The participants' opinions about Workbench Dashboard.

5. Results

The results show that the participants can identify artifacts quickly and accurately. Two participants preferred the interactive visualization in Workbench Dashboard rather than the static visualization created by the W3C RDF Validation Service [12]. The participants also expressed positive and negative feedbacks and suggested potential improvements to the system.

5.1. Identifying UI button and provenance information

Table 3 shows the result of Q1 to Q10. Participant 1 got 9 correct answers. Participant 2 and 3 got all 10 correct answers. The overall correctness is 96.67%, which is high.

Due to an unexpected screen recording problem, the time spent on each question for Participant 1 was not recorded and therefore was shown as N/A in the table. However, according to the recorded time of participant 2 and 3, the overall average time spent on each question is 68.5, which is fast.

Table 4. Result of Q1 to Q10

	Participant 1		Participant 2		Participant 3		Accuracy	Average Time (s)
	Correct	Time (s)	Correct	Time (s)	Correct	Time (s)		
Q1	Yes	N/A	Yes	5	Yes	25	100%	15
Q2	Yes	N/A	Yes	90	Yes	30	100%	60
Q3	Yes	N/A	Yes	120	Yes	330	100%	225
Q4	Yes	N/A	Yes	30	Yes	70	100%	50
Q5	Yes	N/A	Yes	10	Yes	75	100%	42.5
Q6	Yes	N/A	Yes	30	Yes	60	100%	45
Q7	No	N/A	Yes	30	Yes	100	66.67%	65
Q8	Yes	N/A	Yes	25	Yes	20	100%	22.5
Q9	Yes	N/A	Yes	35	Yes	105	100%	70
Q10	Yes	N/A	Yes	30	Yes	150	100%	90
Overall Average:							96.67%	68.5

The results show that the participants could identify the related artifacts and relationships in a short time accurately even though it was the first time they used this application.

5.2. Preference between interactive and non-interactive visualization

Two participants preferred to use Workbench Dashboard rather than the non-interactive graph generated by the W3C RDF Validation Service [12]. One participant stated that the Dashboard's interactive visualization is better when the provenance data is very large. This is because Workbench Dashboard allows users to choose to display or hide some information to avoid overloading of information on the screen while the non-interactive graph loaded all the information in a single picture, which made it difficult to search for information. However, another participant stated that the distribution of text labels in the non-interactive graph is better than those in Workbench Dashboard. The distribution of text labels and nodes need to be improved to avoid moving nodes around frequently to read the text label.

5.3. Positive Feedback

- The overall user interface was intuitive. For example, the comparison between two artifacts could be done by dragging one node to the other node.
- Grabbing and moving things around allowed users to highlight and focus on the elements they are looking for.
- The visualization was appealing and gave users the ability to focus on activities.

5.4. Negative Feedback

- As shown in Figure 24, the commit names were too long so that it used a lot of space on the screen and increased the chance of overlapping with other labels.

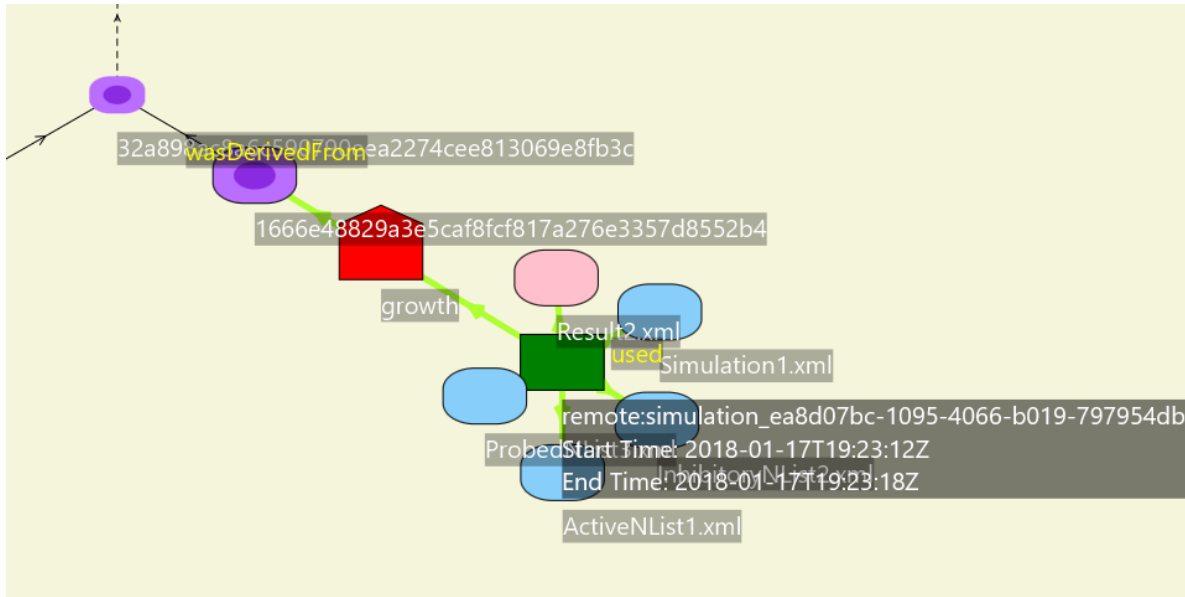


Figure 24. Labels are overlapping with each other although all nodes are displayed on the screen.

- One participant stated that the arrows and labels of the relationships between commit nodes, as shown in Figure 25, were confusing and not intuitive.

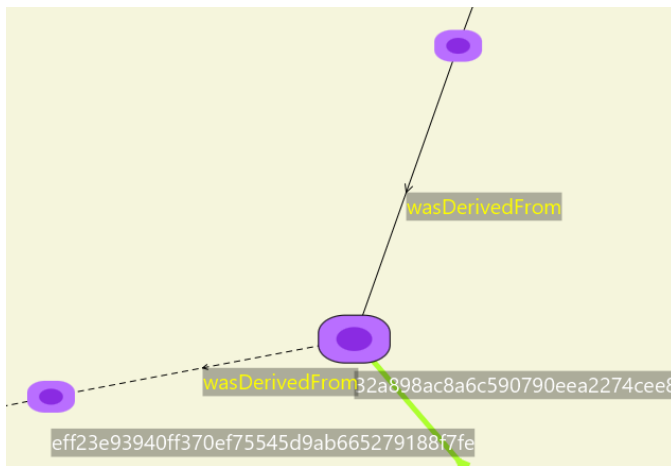


Figure 25. Arrows and labels to show relationships between commits

- Not all the text labels on the edges were helpful. For example, as shown in Figure 15, some users can identify the input and output nodes by using their color without looking at the “used” and “generated” labels. Useless labels may cover the text labels users want to read.

- The spacing between nodes is not optimal. If the spacing is larger, the labels would not overlap with each other, but users could not look at all the nodes in the screen, as shown in Figure 26. If the spacing is small, users could look at all the nodes in the screen, but the labels would overlap with each other, as shown in Figure 24.

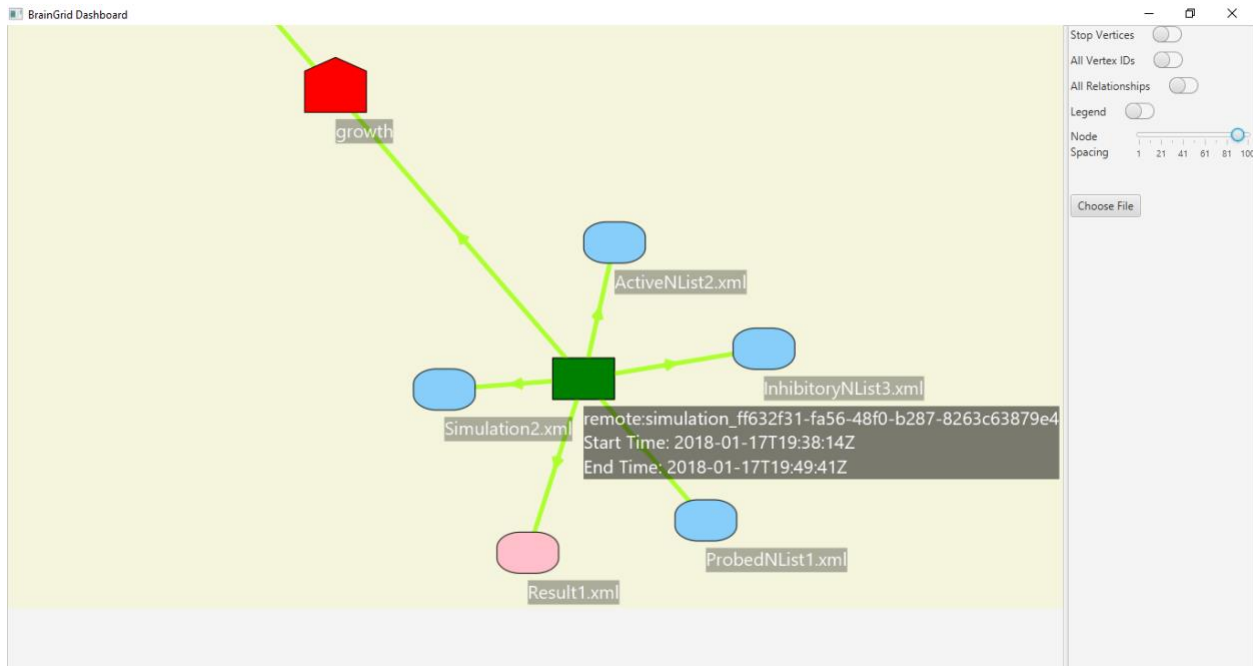


Figure 26. Labels are not overlapping with each other, but the commit node is missing.

5.5. Potential Improvements

- Display shortened commit names by default. Create a control for users to check the complete commit names.
- Improve the algorithm for the distribution of nodes to avoid too many labels overlapping with each other, which makes it difficult for users to read the labels.
- Add a “turn off all labels” button to clear all the highlighted nodes. Therefore, users can clear all labels when the labels in the visualization occupy too much space.
- Add a toggle button to toggle the direction of the arrows of edges and the text labels to make it more intuitive for different people.
- Add a button to turn off the edge labels or make the labels less sensitive to the mouse click so that the labels would not distract users’ attentions.

6. Discussion/Conclusion

This project designed and implemented a software application to help scientists to identify different artifacts involved in the neural simulations created by BrianGrid+Workbench. It employed the interactive visualization of data and software provenance, as well as additional features, like highlighting an activity node and its related nodes, and comparing two artifacts by dragging one node to another node. The usability evaluation shows that the participants learned how to use Workbench Dashboard in a short time. They were given a ten-minute briefing session before answering questions. They identified artifacts in the node-link diagram quickly and correctly. The average time spent on a question is 68.5 seconds. The overall accuracy is 96.67%. User feedback shows that the interactive visualization helped users understand the overview of simulations by showing summary information by default and displaying detailed information when users want to look at more details. The summary information includes artifacts and the connections among artifacts. Artifacts are the nodes in the node-link diagram. Connections among artifacts are the edges among nodes. Detailed information includes artifact names, relationship names and differences among artifacts. Artifact names are the node labels. Relationship names are the edge labels. Differences among artifacts are shown in the side-by-side text view when comparing two artifacts. Participants thought that the overall user interface is quite intuitive. However, there are some negative feedback and potential improvements.

6.1. Future Work

6.1.1. Improvements on the visualization

- Working on the potential improvements collected from the usability evaluation can improve the usability.
- The software provenance information in the visualization is not enough. More details, like bugs and issues related to the development history, need to be added to the visualization.
- Merging artifacts with the same contents can eliminate the number of duplicated nodes and make the graph easier to comprehend.

6.1.2. Improvements on the artifacts comparison feature

- Adding a feature to compare HDF5 files can help the analysis of output files. Currently, the artifacts comparison feature only can compare text files. However, the output files can be in non-text file formats, such as the HDF5 [25].
- Applying machine learning algorithms may assist the analysis of large output files. Depending on the simulation configuration, the size of the output files could be very large, which causes difficulties in visualizing differences among result files. Using machine learning algorithms can potentially discover the pattern of the data in the output files and show insightful differences in the visualization.

6.1.3. Applying Workbench Dashboard to different simulation software

Currently, Workbench Dashboard only visualizes artifacts in simulations created by the BrainGrid+Workbench. However, it may also visualize artifacts in simulations created by other software applications. Visualizing artifacts in other simulations and doing more usability evaluations are beneficial to understanding how to apply Workbench Dashboard to different simulation applications.

7. References

- [1] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. N. d. I. Hidalgo, M. P. B. Vargas, S. Sufi and C. Goble, "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud," *Nucleic Acids Research*, vol. 41, no. 1, pp. 557-561, 2013.
- [2] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao and Y. Zhao, "Scientific workflow management and the Kepler system: Research Articles," *Concurrency and Computation: Practice & Experience - Workflow in Grid Systems*, vol. 18, no. 10, pp. 1039-1065, 2006.
- [3] H. Stitz, S. Luger, M. Streit and N. Gehlenborg, "AVOCADO: Visualization of Workflow-Derived Data Provenance for Reproducible Biomedical Research," *Computer Graphics Forum*, vol. 35, no. 3, pp. 481-490, 2016.
- [4] C. Silva, J. Freire, E. Santos and E. Anderson, "Provenance-Enabled Data Exploration and Visualization with VisTrails," in *SIBGRAPI - Conference on Graphics, Patterns and Images Tutorials*, Gramado, Brazil, 2010.
- [5] M. Stiber, F. Kawasaki, D. Davis, H. Asuncion, J. Lee and D. Boyer, "BrainGrid+Workbench: High-Performance/High-Quality Neural Simulation," in *International Joint Conference on Neural Networks (IJCNN)*, Anchorage, AK, 2017.
- [6] Merriam-Webster, "Provenance|Definition of Provenance by Merriam-Webster," Merriam-Webster, [Online]. Available: <https://www.merriam-webster.com/dictionary/provenance>. [Accessed 13 February 2018].
- [7] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan and J. V. d. Bussche, "The Open Provenance Model core specification (v1.1)," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 743-756, 2011.
- [8] K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo and C. Tilmes, "PROV-DM: The PROV Data Model," World Wide Web Consortium (W3C), 30 April 2013. [Online]. Available: <https://www.w3.org/TR/2013/REC-prov-dm-20130430/>. [Accessed 18 January 2018].
- [9] K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik and J. Zhao, "PROV-O: The PROV Ontology," World Wide Web Consortium (W3C), 30 April 2013. [Online]. Available: <https://www.w3.org/TR/prov-o/>. [Accessed 3 January 2018].

- [10] World Wide Web Consortium (W3C), "RDF 1.1 Concepts and Abstract Syntax," World Wide Web Consortium (W3C), 25 February 2014. [Online]. Available: <https://www.w3.org/TR/rdf11-concepts/>. [Accessed 17 February 2018].
- [11] D. Beckett, T. Berners-Lee, E. Prud'hommeaux and G. Carothers, "RDF 1.1 Turtle," World Wide Web Consortium (W3C), 25 February 2014. [Online]. Available: <https://www.w3.org/TR/turtle/>. [Accessed 21 January 2018].
- [12] World Wide Web Consortium (W3C), "W3C RDF Validation Service," World Wide Web Consortium (W3C), 28 February 2006. [Online]. Available: <https://www.w3.org/RDF/Validator/>. [Accessed 25 January 2018].
- [13] R. Hoekstra and P. Groth, "PROV-O-Viz - Understanding the Role of Activities in Provenance," in *IPAW 2014: Provenance and Annotation of Data and Processes*, Cologne, Germany, 2014.
- [14] I. Suriarachchi, Q. Zhou and B. Plale, "Komadu: A Capture and Visualization System for Scientific Data Provenance," *Journal of Open Research Software*, vol. 3, no. 1, p. e4, 2015.
- [15] Cytoscape Consortium, "What is Cytoscape?," Cytoscape Consortium, [Online]. Available: http://www.cytoscape.org/what_is_cytoscape.html. [Accessed 1 March 2018].
- [16] GitHub, "GitHub," [Online]. Available: <https://github.com>. [Accessed 11 01 2018].
- [17] Oracle, "JavaFX - The Rich Client Platform," Oracle, [Online]. Available: <http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html>. [Accessed 11 01 2018].
- [18] C. Aniszczyk, C. Halstrick, C. Ranger, D. Borowitz, G. Wagenknecht, J. Nieder, K. Sawicki, M. Kinzler, M. Sohn, R. Rosenberg, R. Stocker, S. Zivkov, S. Pearce and S. Lay, "JGit," [Online]. Available: <https://www.eclipse.org/jgit/>. [Accessed 11 01 2018].
- [19] J. Martinez and T. Mikula, "RichTextFX," FXMisc, [Online]. Available: <https://github.com/FXMisc/RichTextFX>. [Accessed 11 01 2018].
- [20] J. Giles, "ControlsFX," [Online]. Available: <http://fxexperience.com/controlsfx/>. [Accessed 11 01 2018].
- [21] D. Naumenko, "DiffUtils," [Online]. Available: <https://code.google.com/archive/p/java-diff-utils/>. [Accessed 11 01 2018].
- [22] E. R.Tufte, in *Envisioning Information*, United State of America, Graphics Press, 1998, p. 58.
- [23] S. G. Kobourov, "Force-Directed Drawing Algorithms," in *Handbook of Graph Drawing and Visualization*, R. Tamassia, Ed., CRC Press, 2016, pp. 385-386.
- [24] F. Kawasaki and M. Stiber, "A simple model of cortical culture growth: burst property dependence on network composition and activity," *Biological Cybernetics*, vol. 108, no. 4, pp. 423-443, 2014.

- [25] The HDF Group, "High Level Introduction to HDF5," 23 September 2016. [Online]. Available: <https://support.hdfgroup.org/HDF5/Tutor/HDF5Intro.pdf>. [Accessed 31 January 2018].
- [26] PROV Working Group, "PROV-Overview," World Wide Web Consortium (W3C), 30 April 2013. [Online]. Available: <https://www.w3.org/TR/prov-overview/>. [Accessed 18 January 2018].
- [27] A. V. Ceguerra, P. V. Liddicoat, S. P. Ringer, W. J. Goscinski and S. Androulakis, "A Tool for Scientific Provenance of Data and Software," in *Computational Science and Engineering (CSE)*, Sydney, NSW, Australia, 2013.
- [28] World Wide Web Consortium (W3C), "SPARQL 1.1 Overview," World Wide Web Consortium (W3C), 21 March 2013. [Online]. Available: <https://www.w3.org/TR/sparql11-overview/>. [Accessed 25 January 2018].
- [29] D. Beckett and T. Berners-Lee, "Turtle - Terse RDF Triple Language," World Wide Web Consortium (W3C), 28 March 2011. [Online]. Available: <https://www.w3.org/TeamSubmission/2011/SUBM-turtle-20110328/>. [Accessed 20 January 2018].