Vrije Universiteit Amsterdam

Bachelor Thesis

Visualisation of Distributed Algorithms Executions: Improving the DaViz Tool

Author: Daniel A. Roos (2550957)

1st supervisor: W. Fokkink
daily supervisor: H.-D. A. Hiep
2nd reader: J. Limperg

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science July 26, 2022

Visualisation of Distributed Algorithms Executions: Improving the DaViz Tool

Daniel Adimas Roos,
Developer
Vrije Universiteit Amsterdam
Bachelor Computer Science
d.a3.roos@student.vu.nl
2550957

***Abstract*—DaViz is an evolving software tool designed to visualise and simulate distributed algorithms. The intended audience is the students and teachers in the Distributed Algorithm course at the Vrije Universiteit Amsterdam. The program, written in Java and Haskell, is functional but still in its early days of development.**

DaViz was developed by Hans-Dieter and the latest iteration was contributed by Wesley Fu-En Geniz Shann. There was a lack of software that could visualise distributed algorithms with customizable network topologies that could aid in teaching students and supporting lectures. DaViz was created and intended to meet that purpose.

DaViz is a multi-window Java application with a dependency on a functional programming language called Haskell. The software is still in its Alpha phase of development and lacks features for both the end-user and developers willing to contribute to the project. An improvement to aid developers in developing was the use of documentation, and for the end-user experience, the introduction of missing core features such as saving and loading would be implemented too. Considering the infancy of the software and room for improvement, there could be ideas and inspiration acquired from a similarly named software also called DaViz developed by the EEA. This, in conjunction with new potential future works, could assist future contributors in further developing this project.

***Index Terms*—Distributed Algorithm, Simulation, Vizualisation, Design Patterns.**

I. Introduction

DaViz is a software written in Java and Haskell aimed at improving the understanding of distributed algorithms by allowing students and teachers to simulate and visualise the computation of different distributed algorithms in any user-defined network topology. The DaViz software prototype was initially developed in 2017 for the Distributed Algorithms course at the Vrije Universiteit Amsterdam[1]. However, its initial version is still an early prototype and lacks usability, adaptability, content, and features for both developers looking to improve the software and end-users such as teachers and students.

A previous technical report written by Shann indicated that he was able to achieve a big step towards improving DaViz's maintainability by adding version control support[2]. With this, it became possible to fulfil established goals and aid its future development by, for example, adding a Javadoc.

This report's purpose is to describe DaViz's background and the sequence of improvements and corrections it has undergone. This would also include a description of Javadoc, which aims to help future development by informing other developers of the purpose of certain code features, such as how these methods, functions, and variables interact with the rest of the program. Lastly, the rest of the report will go into the project's goals, what was achieved, the difficulties faced, and the goals that were not achieved. It will also cover future works, the comparison between my future work and that of the previous report written by Shann, and prioritised suggestions of what the DaViz software needs to achieve its future goals[2, p. 9-11]. The current list of project goals outlines multiple challenges and objectives, including both achievable ones and others that weren't possible due to technical resource or time limitations.

The first goal was to enhance the development experience. This was achieved through Java's built-in documentation with a Javadoc that provided vital information to developers on how elements in the code interacted with each other. The second was the implementation of core features. This was very broad, but certain features such as the save and load feature were missing and was then implemented. The last goal was achieving the future works of past papers, removing frames to be reduced to a single window, and general improvements to aid the end-user experience, such as implementing a counter.

II. Background

A. Background

An algorithm that operates on a distributed system is called a distributed algorithm. A distributed system is made up of numerous separate computers that don't share memory. Using communication networks, each processor has its own memory. A process on one machine talks to another process on another machine to implement communication in networks. A coordinator is necessary for many distributed system algorithms because they carry out tasks that other processes in the system need.It is the lack of a single centralized control process that separates distributed algorithms from conventional algorithms. This lack of centralisation poses some challenges. One of these challenges is understanding the logistics of these distributed

algorithms. There is no oversight of these distributed algorithms because each network has no supervision. However, by simulating the distributed algorithm on a single machine, the user can make scheduling decisions that this control lacks in a real-world scenario[3].

In a distributed algorithm, several computers, also known as nodes, communicate with each other through a communication channel known as a path or an edge. A collection of nodes and edges is known as a network. Using these paths or edges, nodes transfer messages between other nodes, but the manner in which they exchange messages is through a shared algorithm, that is, the distributed algorithm. The algorithm will determine the messages a node sends during each step, how it interprets the messages it receives, when it stops, and what it outputs when it stops.

DaViz aims to tackle the challenge of visualising the logistics of different distributed algorithms. Through visualisation, users will be able to thoroughly follow the distributed algorithm in any network topology. Additionally, by selecting from a diverse range of different distributed algorithms and giving custom choices at each step of the algorithm, it's possible to achieve a deeper understanding of the overall topic of distributed algorithms

B. Origins

DaViz was first made by Hans-Dieter as a project while studying at Vrije Universiteit Amsterdam. The current version of DaViz is a collaboration between Vrije Universiteit Amsterdam and the Centrum Wiskunde en Informatica as a tool for students and teachers. This latest version is different from its older prototype and has more features and content, achieved by Wesley Fu-En Geniz Shann, who brought a vital feature into the development of DaViz: version control and a Git repository. Version control gives DaViz an evolution-like state in which the software incrementally gets more features and content to reach its end goal. This, combined with a Git repository making DaViz open source, allows other institutions outside of Vrije Universiteit Amsterdam to possibly improve and adapt the software to their liking.

III. DaViz

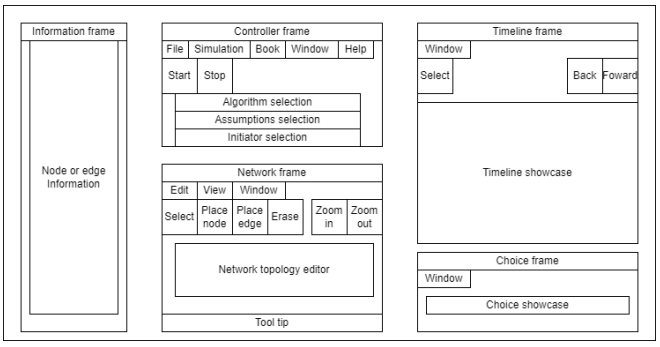

Fig. 1. Layout of the older version of DaViz

DaViz must be initialised by installing a Java SE Runtime Environment (8 and up) for Windows, Mac OS X, and Linux. Alternatively, it is also possible with an open source OpenJDK 1.8.0 on alternative Linux operating systems, such as Debian, Fedora, or Ubuntu. Afterwards, the application can be started by running the designated .jar file.

Once the application has started, there will be four windows: the Information window, the Controller window, the Network window, and the Executions window. The user, at this point, can only interact meaningfully with the Network and Controller windows. This is because the Execution and Information windows are still empty and await the running of the actual simulation before they become relevant. In a normal scenario, the user could make their own network topology using the buttons in the Network window. The possibilities are making nodes and edges, erasing either, and selecting individual elements in the network topology. After the network has been made, a user selects an algorithm from the list in the Controller window. The currently available algorithms are Tarry, Depth-first search, Awerbuch, Cidon, Tree and Echo. Then the application makes an assumption regarding the network topology being acyclic, centralised or decentralised. This feature, however, is not yet implemented as of writing, and the only assumption available is centralised. Next, the user has to select an initiator node from the network topology made in the Network window. Finally, the user can press the "Start Simulation" button in the Controller window.

Assuming the network topology was correctly made, the application will start simulating the desired results. At this point, the user can follow the algorithm's exact logistics and desired path by following the left and right arrows in the Executions window. Selecting any nodes within the Executions window will show additional information in the Information window, including the type of message, the type of process, the number of states, and the number of children, among other details. Last, the user could also change their choices and re-run the algorithm by selecting a previously grayed out button in the Controller window. This window was previously called the Choice window but was combined into the Timeline window to make the current Executions window.

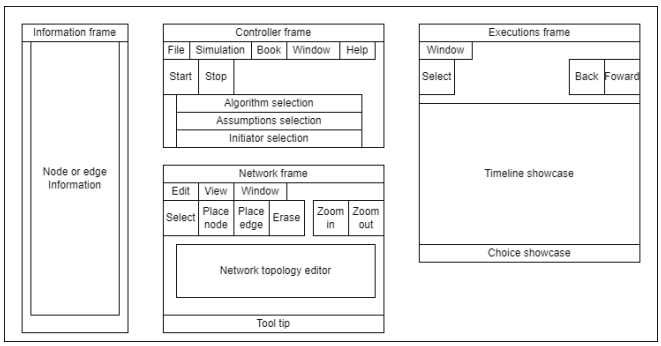

Fig. 2. Layout of the latest version of DaViz

A. Development

To start development and contribute to the software, it is necessary to set up the proper environment first. The

requirements are the Java 8 development kit and the most recent version of Apache Ant. Another requirement is the use of a plugin for Frege, which is a necessary library for the Haskell code. Frege is a Haskell for the JVM. These are the packages present in the DaViz project

These are the packages present in the DaViz project: ——

1) *DaViz:*
2) *DaViz.glue:*
3) *DaViz.glue.alg:*
4) *DaViz.images:*
5) *DaViz.ui:*
6) *DaViz.ui.swing:*
7) *DaViz.ui.swing.plaf:*
8) *DaViz.ui.swing.plaf.basic:*

B. IDEs and Plugins

The project's current build is mainly programmed on the Eclipse IDE (Integrated Development Environment). It is possible to run and build on alternative IDEs with the assumption that the IDE can run Apache Ant build files. Some examples of alternative IDEs are Intellij IDEA and Netbeans. However, it has been noted that future development should be possible on any other IDE. This was specified because of the dependency on third-party plugins for Frege code development. Currently, the third-party plugin for Frege on Intellij IDEA does not appear to be compatible with the code in DaViz[4]. Features such as error detection, documentation viewing, syntax highlighting, and colour schemes are missing, which could prove that development on this code is difficult. This issue is something out of the scope of this project because of its reliance on a third-party plugin that no one can alter except for the plugin's project developer.

To start developing the project on the Eclipse IDE[1], developers are only required to have a version of Eclipse compatible with the Java 8 Development Kit and download the Frege plugin for Eclipse compatible with the project. There is no other third-party plugin required to run the Apache Ant build files. Eclipse should automatically do this as the build file is present within the GitHub repository.

C. Javadoc

Javadoc is a tool for creating HTML-formatted API documentation from source code comments. The API's goal is to provide information about code so that other programmers can use it without having to fully understand how it operates[5]. Javadoc requires specific formatting for comments in order to build the API.

Previous versions of DaViz lacked any form of documentation, which made it difficult for new developers wanting to contribute to this project. Arguably, the most important aspect of working on an existing project is documentation, such as a Javadoc. For example, in the project under the package "DaViz.ui.swing", there exists a class "DefaultGraphModel.Java" with two methods called "addNode(NodeModel n)" and "createNode(float X,float Y)". A new developer could assume that createNode would generate a node and add it to the existing network, and addNode would add an already existing node to the network. However, this is not the case. createNode is part of the DefaultGraphModel that does not interact with the network model and only generates a nodeModel object. It requires being combined with addNode for it to be inserted into the network model. With the presence of a Javadoc, the developer is notified about expectations of methods, variables, and classes before interaction.

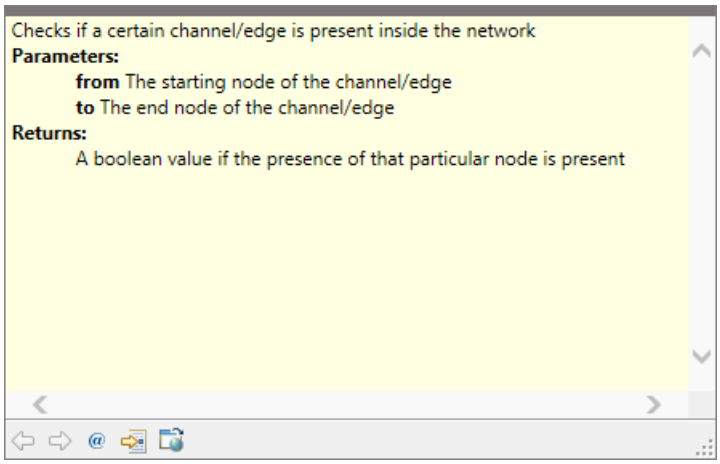

Fig. 3. An example of the presence of a Javadoc within the Eclipse IDE for Network.Java

Furthermore, a Javadoc can be viewed as an HTML document using a Web browser, or it can be viewed dynamically during development on most modern IDEs with the assumption that the IDE supports Javadoc as pop-ups whilst interacting with said method, variable, or class.

The current Javadoc does not have every method, class, and variable documented. This was deliberately done as some methods, variables, and classes that were either redundant or had some static values or purpose would not require changes. On the other hand, methods that dealt with dynamic values that change how the inner workings of the software are run, such as network handling, choice, information, or algorithms, were all extensively documented. However, some static variables were documented, too, for example, node float coordinates (X and Y values).

This documentation is intended to guide developers on the proper method of changing these values, thus reducing the time necessary to understand the code. As is well known, it is usually simpler to create code than it is to comprehend another person's[6]. With the addition of documentation, the time necessary to understand someone else's code is reduced considerably.

D. Current Status

Currently, the project is still in an alpha phase of development as the software still lacks certain features concerning end-user experience. For example, the list of basic distributed

[1]https://www.eclipse.org/ide/

algorithms that can be selected is currently very limited. Shann also noted this in his future works[2].

However, the software now does feature a save and load feature where a user can save any valid network topology to a file of their liking and, if necessary, share and load the same network topology without remaking it entirely from scratch. The addition of this feature was achieved by working on the framework laid out by Shann specifically for this feature and adding extra libraries such as the Java File and Scanner libraries. A variant of the load and save feature is a network saved with custom choices. This is not possible in the current version of the build due to technological and time limitations in the software that need to be addressed.

Next, in terms of UI, the software could be deemed excessive. This is because of the usage of four different windows to display four individual elements of the simulated algorithm. However, this is an improvement over the older five windows as the Timeline and Choice window from the previous version of DaViz are now combined into a single window called the Executions window. Ideally, the number of windows should be limited to one so as not to overwhelm the user with information that may not be relevant at a certain time.

The previous and current versions of the software also still face the issue of dependency on Haskell. To fully comprehend the DaViz implementation, developers would be expected to be knowledgeable with programming in Java and Haskell. Java is a well-known object-oriented programming language, whereas Haskell may not be as common. Given the functional nature of Haskell, it is anticipated that developers unfamiliar with it would encounter certain challenges while beginning the simulation's development. This is further exacerbated by the fact that all development relating to the algorithms would require some knowledge of Haskell and its functionality. For example, it would be impossible to implement any control algorithms without any Haskell knowledge. To alleviate this issue, it would require future development to rewrite the Haskell foundation that DaViz is based on. Furthermore, because of DaViz's dependency on Haskell and, in turn, Frege, as discussed previously, there are limitations to programming environments such as IDEs that new developers who would like to contribute could choose from.

IV. Current Project Goals

The project had several precursor goals set. These goals were determined through discussions with Wan and Hans-Dieter about what would be achievable with the given time frame and technical skills.

A. Improving the development experience

1) Javadoc: With the implementation of the collaboration and version control system via a GitHub repository[6], the next step in aiding development would be the addition of a Javadoc to the rest of the software. This would allow other developers and future contributors to understand methods, variables and classes without needing a deep understanding of the underlying code.

With a Javadoc, this can be all done beforehand and explained in a simple way to future developers so they too can understand the purpose of certain pieces of code.

In Javadoc, a pop-up for a method may look something like this. First, a small description of the method, possibly including hints and a description of what it might not be able to do. Second, parameters written in the Javadoc as "@param" alongside a small description of the expected parameter that the method requires so as not to throw any expectations or errors. For example, a method could require a float. Floats are generally associated with bigger numbers, but the description could state that this is a float for a coordinate on the network topology, giving the developer an idea of the range of the float to input. Third, returns, denoted in the Javadoc as "@returns". This is another short description of the method's output and what could be expected. A method may return a node, but without documentation, it could be an empty node or a node already initialised with variables. Last is the error handling of a method that would be denoted as "@throws". This attaches another small description to show developers what kind of exceptions they could expect if the method throws an exception. Surely this is also done by the actual exception itself, but it would be vital to know which method could throw a certain type of exception.

2) Development environment: Another goal to help improve development is to add more IDE workspaces to the project, as currently, it is only programmable on the Eclipse platform. Early in the project, development started on the IntelliJ IDEA platform and Notepad++ for testing purposes. However, it was quickly determined that the issue regarding different IDE platforms is not the project but rather the third-party plugins necessary to aid in developing the Haskell code, i.e., the Frege plugins. The Eclipse Frege plugin is currently the only one that works consistently with the project. The creator of the Eclipse Frege plugin did not create the plugins for other IDE platforms, meaning that each Frege plugin will behave differently and could possibly be incompatible with the current version of DaViz, depending on the IDE platform it was made on.

B. Implement core features

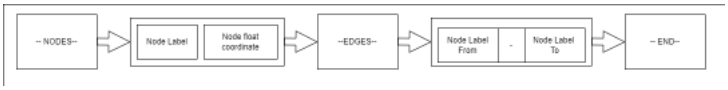

Fig. 4. Layout of a network of the saved file

1) Save and load user defined network topologies: A crucial component absent in DaViz is the ability for end-users to save and load custom network topologies. Previously, any network topology made by the user would automatically be deleted once the application was closed. This would prove to be an issue for users wanting to share the network they had made with other users. It would require users to constantly recreate the same network.

Thus the implementation of this feature was of high priority, and it was also stated as such in Shann's future works in his

technical report[2, p. 9-11]. While there is a certain framework inside DaViz's code, such as the existing code for the selection of save and load buttons, the remainder of the code is entirely blank. The save button is located in the controller menu bar under "File" as "Save Scenario".

The save feature is able to save any network topology after the network has been simulated at least once. The reason for this design choice is that any network simulated through DaViz is automatically checked for a valid provided network. A network could be invalid, for example, if there are not enough nodes or edges present to simulate any algorithm. With the assumption that the network is valid, the user can choose to save the network they generated. Once the user clicks the option to save the network, another window will be generated prompting the user to select the path to the file and the name of the file they wish to save it as.

The file does not have a specific extension for DaViz and, for testing purposes, all files were simply saved as text files ".txt". It was feasible to examine the files' internal workings as text files and confirm that the networks were being preserved correctly. The saved files have certain blocks to delimit them if they were to be scanned for loading into DaViz.Firstly, the file starts with NODES. This is followed by one or two letters to denote the label of the node. Two float values are then present; this determines the coordinates of the node. Secondly, the EDGES block is made. This is followed by one or two letters separated with a dash, and then another one or two letters. This is the edges saved in the file, and the first set of letters determines the source of the edge and the final set of letters determines the destination of the edge. Finally, the file ends with an END block to determine the end of the file.

The load feature placed in the menu bar under "File" as "Load scenario" works by reading a valid file made by the save feature. It functions similarly to the save feature such as a window being generated requesting the user to select a valid file to load into DaViz. Once a file is selected the file is read by scanners from the standard Java scanner library. The first block scanned is the node block which loads all the nodes into the network window of DaViz directly, then the edges block is scanned which loads all the edges in between the nodes that have already been loaded. Lastly, the file is checked for the end of file block to determine the validity of the file. It was thought impossible to have it the other way around since edges cannot be set on non-existent nodes, hence the node data is loaded before the edges data.

An attempt was made to make all classes serializable as it allows the use of the standard Java libraries to save the program's current state. But this was deemed not a viable solution to saving and loading files after many failed attempts because embedded classes such as nodes would not throw exceptions when attempted to be serializable. Additionally, the manner in which the machine would be saved was unknown and would require extensive testing. Given the time frame, it was ultimately abandoned and a more conventional method was chosen.

2) Save and load user defined choices: An amendment to the save and load feature is to give users the ability to save and load custom choices made in the Executions window. This, in combination with the normal save and load feature, would make this core feature complete. However, this addition was not achievable in the scope of time given as its difficulty was much greater than expected. The main issue is that the variable of how choices are saved is not clear. There is no presence of a global variable but rather local built-in variables that are fetched, thus understanding the path which saves the actual custom choices while the program is running is difficult. Another issue is that saving a user's choice requires a large expansion to the save feature as the assumptions, the timeline, the steps, and the choices would all have to be saved as well.

C. Achieve future works of past papers

One goal of this project is to complete some future works of the previous technical report as the DaViz program evolves as a whole. Certain future work laid out in the previous report may be too ambitious for this iteration of the project, such as the implementation of a control algorithm.

1) Fixing choice window: A future work that Shann has stated is "Fixing the Choice Windows in the Java Port"[2, p. 10]. Currently, the version being worked on is the Java port, which is working as intended. The Choice window was also moved into the Timeline window to achieve another one of Shann's future works, which was to reduce the number of windows present.

2) General improvements: Another future work that was within the scope of this project was the general improvements. Although this term is very board Shann goes into detail explaining the general improvements that could be made to the project, such as reducing the number of windows, adding a save and load feature, synchronizing the Timeline window with the Network window, etc[2, p. 10].

There were also feature suggestions, such as the save and load feature, which is mentioned several times outside of the future works, emphasizing the importance of this feature. This goal was achieved and may only require debugging in the future should the need arise. [2]

As mentioned previously, another future work mentioned is reducing the number of windows present. This was also achieved [2].

D. Reducing the amount of windows

The number of windows in previous versions of DaViz was 5. This could prove to be numerous for certain users, also considering that the majority of the windows are not usable or used until a network with an algorithm is simulated. Shann also attempted to remove some windows temporarily but failed to do so. However, there is now success with reducing the number of windows present. The window that has been moved is the choice window. The choice window was combined with the Timeline window from previous versions of DaViz to create the Executions window.

This Execution window has all of the elements of the simulated network once it is run in a single window rather than split between two windows. The Information window may also be a likely candidate to be removed or have appeared later on in the simulation, considering it does not present any information until the simulation is finished. Although the scale of the Information window may prove challenging to have to appear later, it was decided to have it kept in position.

V. Future Works

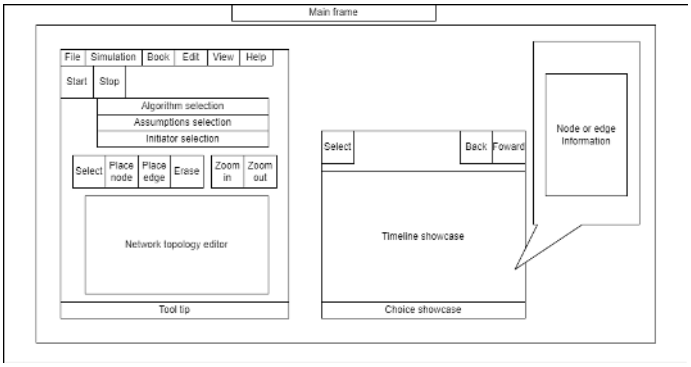

Fig. 5. Concept of an ideal layout

A. Combining all frames into a singular window

Currently, the application starts with 4 frames, which could be deemed excessive by some users, as only 2 out of the 4 are actually in service until an algorithm is simulated. The Controller and Network windows are the only windows the user can interact with in the beginning, as the Executions and Information windows are obsolete. In the most ideal version of the application there would only ever be 1 window which has all the necessary frames such as in the figure above, this would be more intuitive for the users. Even the inactive frames would not appear as a separate entity. Furthermore, for testing purposes, a single window would also reduce the number of visual entities the developers need to test for the front-end of the application[7].

B. Implementation of saving and loading of custom choices

As mentioned in the project goals, currently the save and load features do not have the capacity to save custom-made choices by the user. Future work would include implementing this feature to allow users to save entire custom network topologies, having a specific algorithm simulate the network, and then having a user make different choices, all of which would be saved within a file. the current save and load feature would require further expansion as the foundation has already been set.

C. Allow user-defined algorithms

An interesting suggestion that was also mentioned by Shann[2]. This feature would allow users to define their own algorithms through a custom window. DaViz's current layout already includes some functionality for offering this option[2]. DaViz's event-driven architecture also includes a detailed definition of the algorithm assumptions, events, and data contained within an event. So a user would set the parameters to define their own algorithms. It would also be useful to bring back DaProof for this feature to check the correctness of the user-defined algorithm. So a user would generate their own algorithm with given parameters, save it, and have it simulate a network that the user would also generate. However, before the simulation is processed, DaProof could intuitively check the algorithm beforehand for correctness to see if it is even functional. With this feature alone, it would stand out from other visualization programs.

D. Updating the UI

The current UI is based on Java Swing, and although subjectively it may not look bad, it may look outdated to certain users. A simple future task would be to use a compatible third-party library with DaViz to improve the overall program's visual appearance.

E. Choice in visualization

The Timeline window and Network window currently only have a single form of visualization. Changes to the Network window may prove not necessary as it serves a single purpose. The Timeline window, however, could be made to display the generated timeline differently. Currently, the timeline displays everything in a linear manner on a 2D scale. It could be intuitive for the user if they choose different types of timelines, such as a single linear line timeline or any other alternatives that may be relevant to the user's network and presentation.

An alternative form of visualization, should the user choose to accept it, may be to present the dynamic changes to the network topology in the Network frame during the execution of the algorithm. Some examples of changes that could happen would be the addition of nodes, the removal of nodes or edges. This would intuitively make the network and execution windows more connected and present the overall experience better to the user.

F. Future works of the past

VI. Software Package

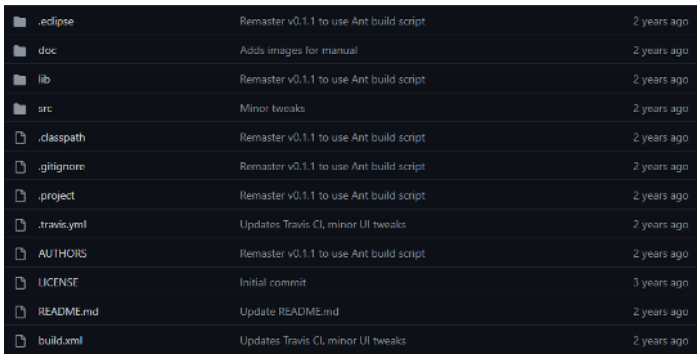

Fig. 6. Contents of the DaViz Git repository

DaViz is accessible to everyone via a GitHub repository. The repository's releases page hosts the executable JAR distribution of DaViz as well as that version's source code. The page includes a README which thoroughly gives a walkthrough to users on how to use the software with visuals[8].

The repository also hosts all the necessary files and libraries to run the program. This includes all the source code of the project, the Apache Ant builder script, and the necessary library to run the Frege code present in the codebase. Thus, any user following the instructions on the README would be able to run the software. However, it should be noted that plugins are not hosted in the repository and that certain IDEs may require additional plugins to start development on the project. Within the repository are.eclipse files, which make the project more compatible with the Eclipse IDE, as other IDEs may not recognize.eclipse package files. Yet, most modern IDEs, such as IntelliJ IDEA, etc., do recognize.eclipse files.

VII. Related Software

A. Comparison of similar software to DaViz

Shann in his technical report discusses in detail several related tools:

Ref	Name	Language	Visualization
[9]	LYDIAN	TCL/TK	X
[10]	IOA	LARCH/C++	-
[11]	DAJ	Java	X
[12]	Khanvilkar	Lydian/C	X
[13]	Mace	C++	X
[14]	Peersim	Java	-

He moreover lists multiple other similar programs to DaViz. While these other programs may share the same intended goal they differ functionally very much from DaViz.

1) LYDIAN: For example LYDIAN is an animation based software to also visualize distributed algorithms with the same intended audience as DaViz, this being students. However, LYDIAN comparatively appears to be less user friendly than DaViz. The LYDIAN software offers no timeline, no custom choices and a very complex menu system mostly used for debugging. Additionally all networks, protocols and animations have to selected in a separate file rather than being coded into the program itself. This however does give LYDIAN an unique functionally of the combination of algorithms being simulated on a network[9].

2) IOA: IOA is a very unique in a sense that it does not attempt to visualize any distributed algorithms but rather built a whole language centered around disturbed algorithms. The language is a functional language and is very much similar to Haskell. This would be logical as the previous and current versions of DaViz heavily rely on the Haskell code. IOA aims to be an input and output automata used to describe and state the properties of distributed algorithms. Due to the software's lack visualization and user friendliness the audience of this program is rather niche[10].

3) DAJ: DAJ is described as an interactive visual aid for studying distributed algorithms. DAJ is more intended for students compared to other applications mentioned before, as it runs the simulations more like interactive video games. By interacting like DaViZ, DAJ has a similar system where users can specify every step of the execution sequence. Although DAJ claims to be a visual aid for students learning distributed algorithms, the software lacks any graphical enhancements and thus purely relies on simple text boxes rather than visualization. Unlike other tools DAJ still sees updates even after 20 years[11].

4) Khanvilkar: Whilst this application has no public software, it was made by Khanvilkar and Shatz and thus will be referred to as Khanviatz. Khanviatz is another application which aims to simulate distributed algorithms. This program is purely written in Lydian. The reason for this choice was described by the writers as that C and Java did not have the proper framework to properly simulate distributed algorithms . However, Khanviatz is a non-visual and purely prototype application. Khanvilkar and Shatz only made a series of models and prototypes to illustrate the possibilities of having an application be able to plot the simulation of a distributed algorithm on a graph and to compare different protocols. These prototypes are very comparable to DaViz's timeline visuals, as this is what Khanvilkar and Shatz also tried to achieve [12].

5) Mace: Mace is not an independent application but rather a language extension and source-to-source compiler that creates a C++ implementation from a brief yet expressive distributed system specification. Comparisons between MACE and DaViz may not be entirely relevant, as MACE is focused on laying a platform and foundation for other applications aiming to simulate and visualize distributed algorithms. It could be, however, interesting to note that an alternative version of DaViz built on MACE could be entirely possible as MACE has numerous abilities to aid in the addition of algorithms and even check the correctness of custom algorithms, something that needs to be separately inspected in DaViz[13].

6) Peersim: Peersim is a peer-to-peer scalable simulator. The project brags about its widespread use in research but appears to have no intention of being used by students. Peersim appears to be rather generic and an functionally similar version of Khanviatz, as it is text based application. The software is command-line based software with minimal visual elements. However, Peersim does have a unique feature called Newscast, which is an epidemic content distribution and topology management protocol[15]. The advantage of DaViz that Peersim presents is scalability. Peersim can run much larger network simulations than DaViz is able to, in combination with its modularity. It can be seen why Peersim is very suitable for research purposes[14].

B. DaViz

Other software which also visualizes data is present on the internet, but this was already discussed by Shann. A point of interest is talking about another software, also named DaViz, made by the EEA, the European Environment Agency.

This DaViz project focuses solely on visualizing data and not distributed algorithms. This implies that the first letters of this DaViz mean ”Data” instead of ”Distributed Algorithms.” The EEA’s DaViz is part of the EEA Design System, which aids different users in specialized design for a specific role.[2]

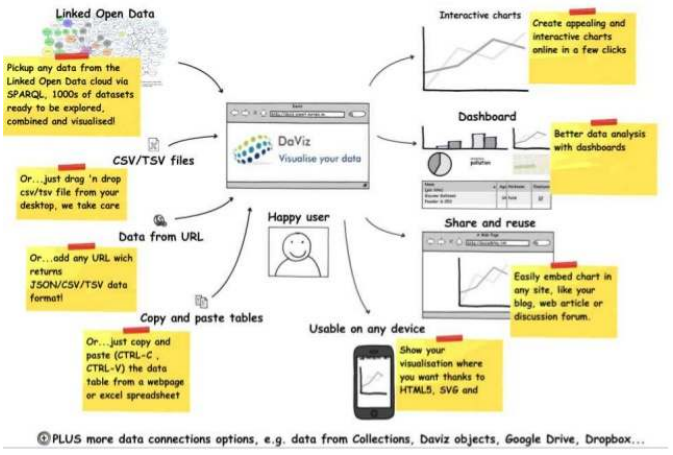

Fig. 7. Layout of features of the EEA DaViz

Even though this project is fundamentally different as it serves a different purpose, it is still possible to compare certain pillars because both projects have an objective of visualization. The DaViz made by the EEA has already achieved what the DaViz of this project is trying to achieve. Some of those achievements are: Firstly, the portability of the software’s development.EEA’s DaViz is written in Python, Javascript and the visuals are done in CSS, these languages are subjectively easier compared to a combination of Java and Haskell, and without the dependency on a third-party plugin it could also allow developers a free choice of their IDE platform.. Secondly, DaViz by EEA has multiple forms of visualization data that can be rendered into charts, graphs, and other visual aspects, whereas the DaViz of this paper only visualizes the timeline. Lastly, sharing and reusing, although this feature may be broad in reference to the save and load feature of the DaViz of this paper, the save and load feature of DaViz by the EEA is very broad in scope as it is able to load files of CSV, TSV, and also load directly from URLs. It also allows the pasting of tables and charts directly to other services such as web pages, web articles or discussion forms, implying a certain amount of ambiguity in the data.

In conclusion, the DaViz by the EEA is much bigger in scope, but this is attributed to the technical capacity of this project compared to the DaViz of this project. However, the projects do align in certain areas due to the fact that both applications aim to visualize something for the user. That is, the EEA may be able to obtain certain learning aspects and ideas from the DaViz.

[2]https://github.com/eea/eea.daviz

VIII. Conclusion

The project has a lot of potential for growth. DaViz currently fills a need that no other software that is fundamentally similar offers. The latest iteration of the project now has extra core features to help the end users; the save and load feature. Although it was planned to further expand this feature to also save user-made choices, this proved too difficult given the scope and time frame as this feature would require more than previously thought extra framework. Other features for the end-user, such as reducing the number of windows and minor bug fixes, have also been achieved. For developers, the addition of a Javadoc was also achieved, but not in full-scope, as it was deemed that certain classes or methods did not require writing. In conclusion, the project has made strides, but it would take more iterations to achieve a fully functional open source software that both users and developers can experience. Lastly, a suggestion considering the equal naming of the project made by the EEA is to rename the project to DiaViz. This name is true to the current name without large changes. The proposed name is also true to the purpose of the project and is easy to remember.

IX. Acknowledgement

I would like to thank Professor Wan Fokkink and Hans-Dieter Hiep. Hans-Dieter, who originally developed DaViz, for all the support he provided in explaining the framework, as well as participating in meetings and giving very helpful feedback. Thanks to Wan for the opportunity to tackle this project, for mentoring a bachelor student with limited knowledge of distributed algorithms and providing resources to help me achieve my goals for this project. And finally, be understanding of the hardships the proposer had to endure during this timeframe in his personal life.

References

[1] H.-D. A. Hiep, “ Vrije Universiteit Amsterdam, Department of Computer Science, Tech,” in *Technical report: Daviz*, 2017.
[2] W. F.-E. G. Shann, “DaViz – Distributed Algorithms Visualization Tool,” in *Individual Systems Practical, 2019-2020 Edition, Technical Report*, 2020.
[3] W. Fokkink, “Distributed Algorithms – An Intuitive Approach, 1st ed,” in *Cambridge, Massachusetts London, England: The MIT Press*, 2013.
[4] Petyo, “https://plugins.jetbrains.com/organizations/IntelliJ-Frege (accessed Jul. 17, 2022),” in *Intelijj’s Frege Plugin*, 2022.
[5] A. Lakhotia, “Understanding someone else’s code: Analysis of experiences,” in *The Center for Advanced Computer Studies University of Southwestern Louisiana*, 1993.
[6] Oracle, “How to Write Doc Comments for the Javadoc Tool — Oracle Nederland, Oracle.com, 2019. https://www.oracle.com/nl/technical-resources/articles/java/javadoc-tool.html (accessed Jul. 16, 2022).” in , 2022.
[7] S. Bresciani and M. J. Eppler, “The Benefits of Synchronous Collaborative Information Visualization: Evidence from an Experimental Evaluation,” in *IEEE Transactions on Visualization and Computer Graphics*, 2009, pp. 1073–1080.
[8] H.-D. A. Hiep, “DaViz - Distributed Algorithms Visualization,” in *https://github.com/praalhans/DaViz (accessed Jul. 17, 2022)*, 2018.
[9] B. Koldehofe, M. Papatriantafilou, and P. Tsigas, “Lydian: An extensible educational animation environment for distributed algorithms,” in *ACM Journal on Educational Resources in Computing*, 2006.
[10] S. J. Garland, N. A. Lynch, and M. Vaziri, “Ioa: A language for specifying, programming, and validating distributed systems,” in *Unpublished manuscript*, 1997.

[11] M. Ben-Ari, “Interactive execution of distributed algorithms,” in *ACM Journal of Educational Resources in Computing,*, 2001, p. 8.
[12] S. Khanvilkar and S. M. Shatz, ““Tool integration for flexible simulation of distributed algorithms,” in *Software: Practice and Experience*, 2001, pp. 1363–1380.
[13] C. E. Killian *et al.*, “Mace: language support for building distributed systems,” in *ACM SIGPLAN Notices*, 2007, pp. 179–188.
[14] A. Montresor and M. Jelasity, “Peersim: A scalable p2p simulator,” in *IEEE Ninth International Conference on Peer-to-Peer Computing*, 2009, pp. 99–100.
[15] M. Jelasity *et al.*, “The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations,” in *Lecture Notes in Computer Science book series*, 2004.