

Qualitative Comparison of Systems Facilitating Data Structure Visualization

Jhilmil Jain, James H. Cross II, and Dean Hendrix
Department of Computer Science and Software Engineering
Auburn University, AL 36849

{jainjhi, cross, hendrix} @ eng.auburn.edu

ABSTRACT

The development of pedagogically sound learning tools using software visualization (SV) techniques has been a very popular area of research. In this paper we will conduct a qualitative comparison of a few such tools with an emphasis on data structure visualization. There are numerous tools available in academia to aid in the instruction of introductory level data structures. In this paper, we will evaluate a representative sample of these tools using Price's SV taxonomy and suggest improvements that can be incorporated in future work.

Categories and Subject Descriptors

K.3.2 [Computers & Education]: Computer & Information Science Education - *Computer Science Education*; E.1 [Data]: Data Structures – *Arrays, Graphs and networks, Lists, stacks, and queues, Trees*

General Terms

Algorithms, Human Factors

Keywords

Taxonomy, data structure visualization systems, qualitative comparison

1. INTRODUCTION

Various taxonomies for SV can be found in the literature, with Myers [5] publishing one of the first in 1990. He suggested classifying systems based on a 2 x 3 grid of aspect vs. display style. Aspect consists of what is being visualized (code, data or algorithm) and display style consists of static or dynamic illustrations. Shu described in her book [9] a classification of SV systems based on what they present (data presentation, program construction and/or execution, software design), and their use as visual coaching systems (systems that bridge the gap between the process of creating a mental model and a program while solving a problem). Singh et.al. [10] published a taxonomy for SV systems very similar to Myers. They use aspect and form for classification purposes. Stasko and Patterson [11] used four measures – aspect, abstraction,

animation, and automation. Stasko and Kraemar [4] classified systems using two dimensions - visualization task (such as data collection, data analysis, storage, display), and purpose (such as debugging, performance evaluation, program visualization). Brown [12] used three measures: content (direct, synthetic), persistence (current, history), and transformation (incremental or discrete). Roman and Cox [8] used five classification dimensions - scope, abstraction, specification method, interface and presentation. In 1993, Price et.al. [7, 12] published a comprehensive taxonomy. This seems to be the most complete taxonomy we have found in our research, and we will be using Price's nomenclature to classify data structure visualization systems.

2. SYSTEMS

This section gives an overview of the six data structure visualization systems that we consider in this paper. In our research we found that these systems are a representative sample of tools used in introductory level data structures and algorithms courses.

ANIMAL: A New Interactive Modeller for Animations in Lectures [14] is a system for creating algorithm and data structure visualizations using a visual editor or scripting commands. Using the editor, novice users can generate or edit animations visually without using any programming code. Objects such as points, polygon/polylines, text, list elements, and arcs can be added to the animation using drag and drop. Advanced users can also use ANIMAL's scripting language for creating animations. Using this tool, animations can be displayed using video-player like features such as play, pause, rewind, or jump to a given step. Source code or pseudo code and textual descriptions can be embedded within the animation. The system's flexibility does not restrict it to introductory computer science courses, and also provides platform independence.

JAWAA: The Java And Web based Algorithm Animation [1, 6, 13] is a scripting language that facilitates easy creation of web-based animations. General-purpose animations as well as data structure animations can be created in a matter of minutes. First, a .anim file containing JAWAA commands or scripts is created by hand or by using the JAWAA editor. The JAWAA editor allows creation of animations using a GUI by laying out objects. This .anim text file is then called as an applet from an html web page to generate animations on the web. JAWAA is language independent and no prior programming experience is required to use it.

jGRASP Object Viewers [3]: jGRASP is a lightweight development environment created specifically to provide automatic generation of software visualizations for the purpose

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
43rd ACM Southeast Conference, March 18-20, 2005, Kennesaw, GA, USA. Copyright 2005 ACM 1-59593-059-0/05/0003...\$5.00.

of improving the comprehensibility of software. In order to generate data structure viewers, a program must run in the jGRASP integrated Java debugger or from the jGRASP object workbench. For any data structure class to be visualized, an “external viewer” is first created using the flexible graph drawing language (FLGL). FLGL is a jGRASP internal graph drawing library that is used for the construction, display, and layout of graphs. Multiple synchronized and dynamically generated data structure views of varying degrees of detail are also available. A graphical interface is also being developed such that additional data structure viewers can be built interactively without using any programming code.

JIVE: The *Java Interactive software Visualization Environment* [15] is a highly interactive system for automatically creating visualizations of programs using its library of pre-coded animated data structures such as graphs, hashtables, and search trees. The graphs and binary search trees are based on the JDLS library. Users can also create stand-alone Java applets with interactive GUIs. JIVE provides an excellent interface for visualizing large data sets using an innovative zooming graphical framework. It also provides a multi-user distributed learning environment such that teachers and students can interact with the same animation or data structure synchronously.

JSAVE: The *Java Simple Automated Visualization Environment* [16] is an interactive system for the visualization of Java Collection classes. Currently, only the List interface is supported. It provides a library of classes that can be directly used in Java programs or XML scripts can be written for visualization purposes. The specialty of JSAVE is the flexibility of user interaction in terms of excellent user control of color, navigation, and multiple representations of the data structure visualizations. Dynamic color customization of components is possible while interacting with the visualizations. The user can play the visualization as a movie, or step through it. JSAVE also allows rewinding the visualization or stepping back through it. The user can dynamically switch between singly linked list, circular list, array, and relative comparison representations as the visualization is running in order to compare the data structures. The ultimate goal of JSAVE is to provide a complete visualization of the functionality of the Java Collection classes.






LIVE: The *Language-Independent Visualization Environment* [2] is a system that enables visualization and manipulation of programs and data structures for multiple languages such as subset of Java, C++, and ÜberLanguage (in-house Pascal like language). The GUI of LIVE consists of two main components: a canvas (on the left hand side) and a source code area (on the right hand side). The user can enter and edit code in the source

code panel. When the code “Runs”, LIVE parses the program, creates a syntax tree, and generates the animation automatically. Since animations are created by interpreting the syntax tree, the user can switch between various code modes, thus allowing the user to view the same code in the syntax of multiple languages. The user can also directly and dynamically manipulate data structures displayed on the canvas and generate source code statements for the same. LIVE is especially useful in understanding the concepts of pointers, linked structures, recursion and effects of the scope of nested variables.

3. COMPARISON DETAILS

We will be using the symbols given in Table 1 to evaluate the features in each category of Price’s taxonomy.


Table 1: Symbols and meaning

<i>Symbol</i>					
<i>Meaning</i>	No or Lowest	Below Average	Average	Above Average	Yes or Highest

3.1 Scope

In this category we have chosen seven features to describe broadly the range of programs that serve as an input (see Table 2). The field “developed in” states the location of the research. The field “URL” specifies the location from where the system can be downloaded by the reader. The “generality” field answers the question – can the system generate visualizations for a generalized range of programs or does it display a fixed (not very flexible) set of programs? It was observed that of the six systems, LIVE and JSAVE were fixed and the others were generalized. The “operating systems” field lists the various platforms supported by the system. All the systems that we assessed supported Windows, Mac, Unix, and Linux. The “programming language of user programs” field lists the type of programming language(s) used by user programs. LIVE was the only system that supports multiple languages. In the field “concurrency support”, we assess if the system can visualize concurrent aspects (if the programming language of the user programs supports concurrency). It was observed that except jGRASP none of the other systems supported any form of concurrent programming visualization. In jGRASP various threads running in the program can be visualized using the debugger. The user has the ability to pause and restart any thread. Lastly, the “specialty” field lists what kind of programs the system is especially good at visualizing. Most systems are well developed in certain specialized areas, but are prototype systems and are not able to handle large datasets.

Table 2: Assessing the “Scope” of the Systems

<i>(1) SCOPE</i>	<i>Developed In</i>	<i>URL</i>	<i>Generality</i>	<i>Operating Systems</i>	<i>Programming Language of User Programs</i>	<i>Concurrency Support</i>	<i>Specialty</i>
ANIMAL	University of Siegen, Germany	http://www.animal.ahrgr.de/		Windows, Mac, Unix, Linux	Animal Script	N/A	Algorithm animation

JAWAA	Duke University, USA	http://www.cs.duke.edu/csed/jawaa2		Windows, Mac, Unix, Linux	JAWAA Script	N/A	Data structure animation
jGRASP Object Viewers	Auburn University, USA	http://jgrasp.org/		Windows, Mac, Unix, Linux	Java		Program and data visualization
JIVE	University of Roma "Tor Vergata", Italy	http://jive.dia.unisa.it/index.html		Windows, Mac, Unix, Linux	Java		Algorithm animation, zooming
JSAVE	Hope College, USA	http://www.cs.hope.edu/jsave/		Windows, Mac, Unix, Linux	Java, XML		Algorithm visualization
LIVE	Hamilton College, USA	http://big-oh.cs.hamilton.edu/~alistair/LIVE/		Windows, Mac, Unix, Linux	subset of Java, C++, ÜberLanguage		Pointers, linked structures, recursion visualization

3.2 Content

In this category, we specifically disregard the fields associated with algorithm visualization since we are concentrating on systems for data structure visualization (see Table 3). The “program code visualization” field assesses the ability of the system to visualize the program source code. jGRASP and LIVE were the only systems that allowed the user to observe the visualizations and source code simultaneously. jGRASP has various features such as the control structure diagram (CSD); the UML class dependency diagram; and a graphical debugger that support program code scalability. In the field “program data flow visualization” we answer how well does the system visualize the flow of data in the program source code? Most systems except jGRASP handle the data flow very poorly. The jGRASP debugger has call stacks to represent data flow. The field “fidelity and completeness” answers the question - does the visualization system present the true behavior of the underlying virtual machine or how closely do the visualizations represent the actual data structures? Hand-designed systems such as ANIMAL and JAWAA are difficult to rank because they depend so much on the individual visualizer. JSAVE and LIVE create

visualizations that are easier to understand, but do not necessarily reflect the underlying virtual machine. jGRASP and JIVE have highest values since they are closely tied to the program code. The field “data gathering time” lists when the data (used in the visualization) is gathered. The field “temporal control mapping” indicates the mapping between “program time” and “visualization time”. All the systems we evaluated had a dynamic-dynamic mapping – i.e. is the data used in the visualization was gathered over a period of time during the program's execution to generate an animation. The field “visualization generation time” lists whether the visualization is created from data gathered in the previous run (post-mortem) or from data produced dynamically as the program executes (live). Systems designed for classroom teaching do not make the connection between the source code used for data structure implementation and the algorithm-level visualization. Though such systems are useful for classroom demonstration purposes, they offer no help to perform lab exercises or assignments. We need a system that is flexible enough to be used in both settings so that students do not have to cope with a number of different tools.

Table 3: Assessing the “Content” of the Systems

(2)CONTENT	Program Code Visualization	Program Data Flow Visualization	Fidelity and Completeness	Data Gathering Time	Temporal Control Mapping	Visualization Generation Time
ANIMAL			Depends on individual visualizer	Run time	dynamic-dynamic	Post-mortem
JAWAA			Depends on individual visualizer	Run time	dynamic-dynamic	Post-mortem
jGRASP Object Viewers				Compile and run time	dynamic-dynamic	Live
JIVE				Compile and run time	dynamic-dynamic	Live
JSAVE				Compile and run time	dynamic-dynamic	Live and post-mortem
LIVE				Compile and	dynamic-	Live

				run time	dynamic
--	--	--	--	----------	---------

3.3 Form

In this category, we evaluated the characteristics of the display or the output of the visualization (see Table 4). The field “color” measures how well the system uses color for effective visualization. If data gathered is at run time, the field “animations” answers the question - how well does the system use animation? All the tools evaluated used color and animations somewhat effectively. The field “dimensions” lists the dimensions a system uses for generating visualizations and “sound” assesses how well the system uses sound to convey information. Almost all of the systems that we evaluated did not explore the benefits of sound and multi-dimensional visualizations. The field “granularity” assesses if the user can manipulate or switch between degrees of detail of the visualization. Most tools were not able to allow the user to switch between the various levels of granularity in order to show/hide data complexities. jGRASP and JIVE (to a certain extent) offer this feature. The field “multiple views” reports whether the system can provide multiple synchronized views (of varying granularity) of the data structures or not. It was observed that most systems did not offer multiple synchronized views of a particular data structure. jGRASP and JSAVE were the only two systems that do offer multiple views. jGRASP is unique in the sense that it offers four views simultaneously – the source code; the low-level object view as seen by the virtual machine; the pedagogical picture of the data structure (example red-black tree); and the high-level view (example sorted list maintained by the red-black tree). Lastly, the field “program synchronization” indicated whether the user can visualize multiple programs simultaneously. Except for jGRASP no other system had this feature. In jGRASP, multiple objects can be created on the workbench and visualized simultaneously.

3.4 Method

In this category, we evaluated the features the system uses to create visualizations (see Table 5). The field “visualization specification style” lists the methods used to produce the visualizations. It was observed that the visualizations had to be hand-coded for four out of six of the systems evaluated. For both ANIMAL and JAWAA, the visualizations have to be hand-coded, but both also provide a graphical interface such that visualizations can be built interactively instead of using programming language/code. This is very useful for novice students who have no prior programming experience. In jGRASP, once an external viewer is created, it is automatically added to the viewer list for that class. The user can then open this viewer on an instance of the class during a debug or a workbench session. JIVE and JSAVE both use a library of classes which must be manually included in the user programs. jGRASP, JSAVE, and JIVE were the only three systems that allow the user to re-use code for visualizations. The field “intelligence” measures if the system uses advanced AI techniques for visualization. Most systems evaluated do not explore this area. The field “tailorability” assesses if the user can customize the visualizations, and if it can, the field “customizable language” specifies the methods used. JIVE allows the user to change the shape of nodes and JSAVE allows the user to change the color and shape during interaction. If a system is not automatic, the field “code ignorance allowance” measures how much knowledge of program code is required for visualization generation. ANIMAL and JAWAA have very low code ignorance, since the user must change the program code to change visualization. jGRASP and LIVE have high code ignorance since visualizations are automatically generated.

Table 4: Assessing the “Form” of the Systems

(3) FORM	Color	Dimensions	Animations	Sound	Granularity	Multiple Views	Program Synchronization
ANIMAL		2D					
JAWAA		2D					
jGRASP Object Viewers		2D					
JIVE		2D with zooming					
JSAVE		2D					
LIVE		2D					

3.5 Interaction

In this category, we evaluate how the user controls and communicates with the system (see Table 6). The field “style” lists the methods used by the user to interact with the system. The majority of tools have buttons or menus to interact with the visualization, but JIVE and JGRASP are the only ones that allow the user to enter data sets while dynamically interacting

with the visualizations. The field “navigation” assesses how effectively the system displays visualizations of very large datasets. Except for JIVE, most systems are not able to display large data sets effectively. JIVE has a unique zooming interface that allows the user to zoom in and out without a loss in resolution. The field “elision control” measures if the user can suppress information/detail from the display. The ability of the user to reverse or rewind the visualization and control the speed

of the visualization are indicated by the fields “temporal control of direction” and “temporal control of speed”.

Table 5: Assessing the “Method” of the Systems

<i>(4) METHOD</i>	<i>Visualization Specification Style</i>	<i>Intelligence</i>	<i>Tailorability</i>	<i>Customizable Language</i>	<i>Code Ignorance Allowance</i>
<i>ANIMAL</i>	Hand-coded, interactive			procedural	
<i>JAWAA</i>	Hand-coded, interactive			procedural	
<i>jGRASP Object Viewers</i>	Automatic, interactive			procedural, interactive manipulation	
<i>JIVE</i>	Hand-coded, library			procedural, interactive manipulation	
<i>JSAVE</i>	Hand-coded, library			interactive manipulation	
<i>LIVE</i>	Automatic			procedural, interactive manipulation	

Most systems except ANIMAL and JSAVE do not allow the user to step back or rewind. This is a very useful feature that a student can use to compare the state of the data structure before and after applying an operation. Animal and JSAVE offer absolute control over the visualization speed – it can be played as a movie (speed of that can be adjusted) or the user can step through. JAWAA also allows the user to step through or play it as a movie but it offers no control over the speed of the movie. jGRASP and LIVE allow the user to step through the code and view the visualization dynamically. JIVE on the other hand does not allow the step through feature, but the animation can be paused and played. The field “scripting facilities” indicated if the interactions with the visualization can be recorded and played back. Most systems lack in scripting facilities. LIVE

offers very basic features to store interactions with the visualization – no other system has this ability.

3.6 Effectiveness

This category is a very subjective measure (see Table 7). The field “purpose” lists the intended goals of the system. The field “experimental evaluation” shows that none of the systems assessed were evaluated empirically. The field “production use” shows that all the systems that we chose are used in an academic environment. We have observed that even though systems have been in use for a long time they have not being evaluated for their effectiveness. This is one area of software visualization that calls for further research.

Table 6: Assessing the “Interaction” of the Systems

<i>(5) INTERACTION</i>	<i>Style</i>	<i>Navigation</i>	<i>Elision Control</i>	<i>Temporal Control of Direction</i>	<i>Temporal Control of Speed</i>	<i>Scripting Facilities</i>
<i>ANIMAL</i>	buttons					
<i>JAWAA</i>	buttons					
<i>jGRASP Object Viewers</i>	buttons, menus, text box					
<i>JIVE</i>	buttons, menus, text box					
<i>JSAVE</i>	buttons					
<i>LIVE</i>	buttons, menus					

4. CONCLUSIONS







It would be ideal to have a data structure visualization system that serves the dual purpose of a classroom demonstration tool and a development environment to be used for lab exercises and assignments. This way, students and instructors will not have to deal with a number of different tools. Future systems should

enable visualization of concurrent programming features, multiple synchronized views of data structures, and program synchronization. It would also be useful to explore the benefits of features such as sound and multi-dimensional rendering. The ability to save the interactions with visualizations for future playback would aid students in revisiting material covered in class. The user should be able to visualize large data sets and

trace program data flow. It would be useful if students had full

control over the speed and direction of the visualization.

Table 7: Assessing the “Effectiveness” of the Systems

(6) EFFECTIVENESS	Purpose	Experimental Evaluation	Production Use
ANIMAL	novice and expert classroom demonstration		Academic (1998)
JAWAA	novice classroom demonstration		Academic (1998)
jGRASP Object Viewers	novice and expert classroom demonstration, development and debugging		Academic (2004)
JIVE	novice and expert demonstration and algorithm development (local and remote)		Academic (2002)
JSAVE	novice classroom demonstration		Academic (2003)
LIVE	novice classroom demonstration and development		Academic (2002)

Lastly, empirical evaluations must be carried out to gauge the effectiveness of data structure visualization tools.

The work presented in this paper is part of a project in progress. The comparisons presented are of a subjective nature. Subsequent work will be directed toward the collection of sufficient data to enable us to quantify the results using statistical metrics.

5. REFERENCES

- [1] Akingbade, A., Finley T., Jackson D., Patel P., and Rodger S. H., JAWAA: easy web-based animation from CS 0 to advanced CS courses, *Proceedings of the 34th technical symposium on Computer science education (SIGCSE)*, February 19-23, 2003, Reno, Nevada, USA.
- [2] Campbell A. E. R., Catto G.L., and Hansen E. E., Language-independent interactive data visualization, *Proceedings of the 34th technical symposium on Computer science education (SIGCSE)*, February 19-23, 2003, Reno, Nevada, USA.
- [3] Hendrix D.T., Cross J.H., and Barowski L.A., An extensible framework for providing dynamic data structure visualizations in a lightweight IDE, *Proceedings of the 35th SIGCSE technical symposium on Computer science education (SIGCSE) 2004*, pp.387-391.
- [4] Kraemer E. and Stasko J., The visualization of parallel systems: an overview, *Journal of Parallel and Distributed Computing*, vol. 18, no. 2, June 1993, pp. 105-117.
- [5] Myers, B.A., Taxonomies of Visual Programming and Program Visualization, *Journal of Visual Languages and Computing*, vol. 1, no. 1, 1990, pp. 97-123.
- [6] Pierson W.C., and Rodger, S.H., Web-based animation of data structures using JAWAA, *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education (SIGCSE) 1998*, pp.267 - 271.
- [7] Price, B.A., Baecker, R.M., and Small, I.S., A Principled Taxonomy of Software Visualization, *Journal of Visual Languages and Computing*, vol. 4, no. 3, 1993, pp. 211-266.
- [8] Roman, G.-C., and Cox, K. C., A taxonomy of program visualization systems, *IEEE Computer*, vol. 26, no. 12, pp. 11-24, December 1993.
- [9] Shu N. C., *Visual programming*, Van Nostrand Reinhold Co., New York, NY, 1988.
- [10] Singh G. and Chignell M.H., Components of the visual computer: a review of relevant technologies, *Visual Computer* vol. 9, issue 3, November 1992, pp. 115-142.
- [11] Stasko, J. and Patterson, C., Understanding and Characterizing Software Visualization Systems, *Proceedings of the 1992 IEEE International Workshop on Visual Languages*, September 1992, pp. 3-10.
- [12] Stasko, J., Brown M.H., and Price B. A., *Software Visualization*, MIT Press, Cambridge, MA, 1997.
- [13] Rodger, S.H., Introducing computer science through animation and virtual worlds, *ACM SIGCSE Bulletin*, vol.34, no.1, March 2002.
- [14] Rößling, G., Freisleben, B., ANIMAL: A System for Supporting Multiple Roles in Algorithm Animation, *Journal of Visual Languages and Computing*, vol. 13, no. 3, pp. 341-354, Elsevier, Amsterdam, The Netherlands, 2002.
- [15] JIVE. Available at <http://jive.dia.unisa.it/>
- [16] JSAVE. Available at <http://www.cs.hope.edu/jsave/>