Extensible and Dynamic Data Structure Viewers in Java

Jhilmil Jain
Computer Science and Software Engineering Department,
Auburn University, Auburn AL
Email: jainjhi@auburn.edu

Problem & motivation

Many techniques for the visualization of data structures and algorithms have been proposed and shown to be pedagogically effective. Yet, they are not widely adopted because they lack suitable methods for automatically generating the visualizations, lack integration among visualizations, and lack integration with basic integrated development environment (IDE) support. In this work, additionally it was identified that the lack of adoption was because these tools do not focus on one of the main problems students in an introductory level data structures and algorithms class face; that is, the transition from abstract and static concepts to dynamic implementation.

All computer science, software engineering, computer engineering, and wireless engineering (software option) majors at Auburn University are required to take COMP 1210 Fundamentals of Computing I. COMP 1210 provides an introduction to the Java programming language. This course is followed by COMP 2210 Fundamentals of Computing II, which is the introductory level data structure course. It uses an object-oriented approach to introduce the basic concepts, design, implementation and application of fundamental data structures.

Data structures and algorithms are abstract concepts, and the understanding of these topics and the material covered in class can be divided into three levels: a) Conceptual – where students learn concepts of operations such as create, add, delete, sort etc; b) Coding – where students implement the data structure and its operations using any programming language (Java in this case); and c) Application - where students choose the most appropriate data structure to solve a programming exercise. Over the course of the past few years a consistent decline in enrollment in the computer science department has been observed. This trend is most noticeable during COMP 2210 when quite a few students decide to drop this required course. Paper-based surveys and one-on-one interviews were conducted in Fall 2004 and Spring 2005 to understand the aspects of the COMP 2210 course that students find most difficult. It was determined that students did not find fundamental concepts difficult to understand but had the most trouble with the implementation. About 75% of students indicated that they had an appropriate level of expertise in Java to complete the requirements of COMP 2210. Hence, poor Java skills may not be causing the problems with implementation. Most students faced a blank-screen syndrome when they began implementation. The basic problem is that students have difficulty transitioning from static textbook concepts to dynamic programming implementation. Thus, there is a need to bridge the gap between concepts and implementation [Jain et al. 2005a].

Background & related work

Over 21 tools that are used for the purpose of data structure visualization were surveyed [Jain et al. 2005b] and it was found that most tools (more than 14 in the survey) focused on conceptual understanding. Only seven implementation level tools included in the survey were intended to help students during program comprehension and debugging activities. But, none of these implementation tools fulfilled all of the following research goals:

Pedagogical goals:

1. Actively engage students.
2. Reduce cognitive load of the short term memory so that efforts can be directed to problem solving.
3. Easy transition from static textbook concepts to dynamic implementation.

4. Reduce number of tools required by using one tool that serves the dual purpose of classroom demonstration and development environment.

Design goals:
1. Provide automatic generation of views.
2. Provide multiple and synchronized views.
3. Provide full control over the speed of the visualization.

The approach we have taken for the data structure viewers in jGRASP is to automatically generate the visualization from the user's executing program and then to dynamically update it as the user steps through the source code in either debug or workbench mode. This is somewhat similar to the method used in Jeliot [Kannusmaki et al. 2004]. However, jGRASP differs significantly from Jeliot in its target audience. Whereas Jeliot focuses on beginning concepts such as expression evaluation and assignment of variables, jGRASP includes visualizations for more complex structures such as linked lists and trees. In this respect, jGRASP is similar to DDD [Zeller 2001]. The data structure visualization in DDD shows each object with its fields and shows field pointers and reference edges in a general way that is not tailored to the type of data structure being viewed. In jGRASP, each category of data structure (e.g., linked list vs. binary tree) has its own set of views and subviews which are intended to be similar to those found in textbooks. Although we are planning to add a general linked structure view, we began with the more intuitive "textbook" views to provide the best opportunity for improving the comprehensibility of data structures.

We have specifically avoided basing the visualizations in jGRASP on a scripting language, which is a common approach for algorithm visualization systems such as JHAVÉ [Naps 2005]. We also decided against modifying the user's source code as is required by systems such as LJV [Hamer 2004]. Our philosophy is that for visualizations to have the most impact on program understanding, they must be generated as needed from the user's actual program during routine development.

Uniqueness of the approach

The jGRASP IDE has been extended to include new dynamic viewers specifically intended to generate traditional abstract views of data structures such as linked lists and binary trees. The purpose of these viewers is to provide fine-grained support for understanding instances of classes representing data structures. When a class has more than one view associated with it, the user can have multiple viewers open on the same object with a separate view in each viewer. These viewers are tightly integrated with the jGRASP workbench and debugger.

The purpose of the viewers is to aid in the understanding of the data structures themselves and to assist in finding errors while developing a data structure. To further this intended use, any local variables of the structure's node type are also displayed, along with the links between these local variable nodes or structure fragments and the main data structure. This allows mechanisms of the data structure such as finding, adding, moving, and removing elements to be examined in detail by stepping through the code. As an additional aid to understanding the mechanisms of the data structure, structural changes are animated in the viewers.

Initially, jGRASP viewers could only be generated using an API based approach. Source code for example viewers that use the API is included with the jGRASP distribution to expedite the creation of new viewers by students and/or faculty. Although a new viewer can be created by changing about 10 lines of source code in one of the examples, this approach proved somewhat impractical for the general CS2 population [Hendrix et al. 2004]. While this option needs to be available for faculty, it was soon discovered that it was unrealistic to expect students who are in the process of learning about data structures to be also able to modify a separate viewer class in order to see an instance of their own data structure. Research efforts were thus directed towards building a mechanism that could determine if an instance was a linked list or binary tree based on a class structures and a set of heuristics, and then automatically generate an appropriate view [Hendrix et al. 2006].

An Example

BinaryTreeExample.java, which is provided with jGRASP, is intended to be representative of a "textbook" example or of what a student may write. Its main method creates an instance of BinaryTree and then adds instances of BinaryTreeNode to it. The UML class diagram in Figure 1 shows that BinaryTreeExample depends on BinaryTree which depends on BinaryTreeNode.

In order to open a viewer, a breakpoint is set in main, and then the program is run in debug mode. After an instance of BinaryTree is created, a viewer is opened by dragging the instance out of the debug window. During the process of opening the viewer, the Data Structure Identifier determines, in this case, that the object is a binary tree structure and opens the appropriate viewer. As the user steps though the program and into the add() method, the nodes appear in the viewer. Figure 2 shows the program while stepping in the add() method. Figure 3 shows the instance of BinaryTree after three nodes have been added and a fourth node is about to be added. Local variable branch indicates the position in the tree where the fourth node will be added. When the fourth node is added, the animation provided by the viewer shows the node "sliding" up into the tree. Figure 4 depicts the viewer after the node has been added but prior to size being incremented. Notice that size is incremented just below the location of the debug step in Figure 2. Students have indicated that seeing the links being set correctly (or incorrectly) as they step through their code is extremely helpful with respect to their understanding of exactly how the implementation relates to the abstraction of the data structure itself. That is, seeing a node added on the blackboard as links are redirected is easy, but when it comes to understanding how this happen in the actual code, it is suddenly not so easy. However, seeing the data structure updated in the viewer as individual statements are executed makes a direct connection between the implementation and the abstraction, and therefore provides a greater opportunity for deeper understanding [Cross et al. 2007].

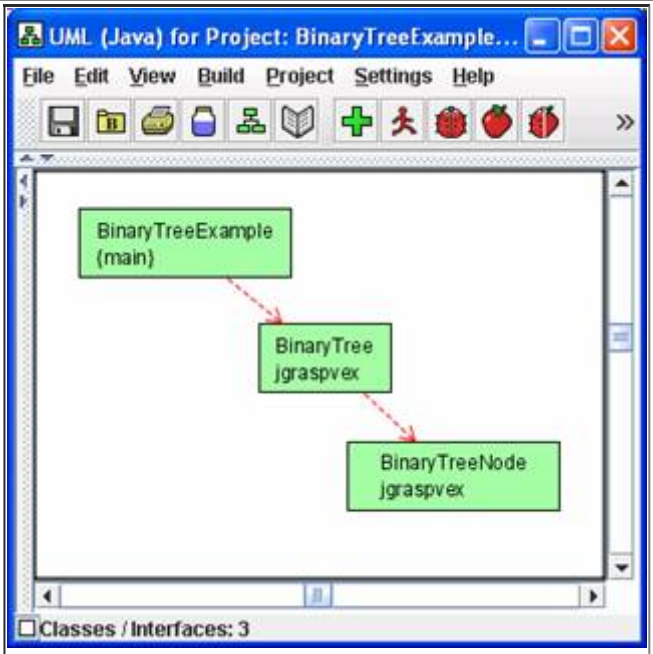

Figure 1. UML class diagram of BinaryTreeExample

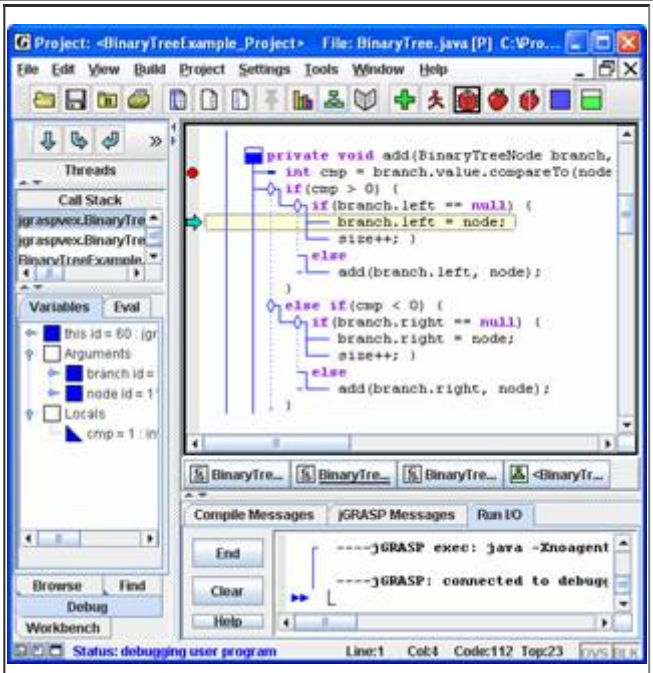

Figure 2. CSD window of jGRASP with the debugger after stopping in at a break point and then stepping in the add() method

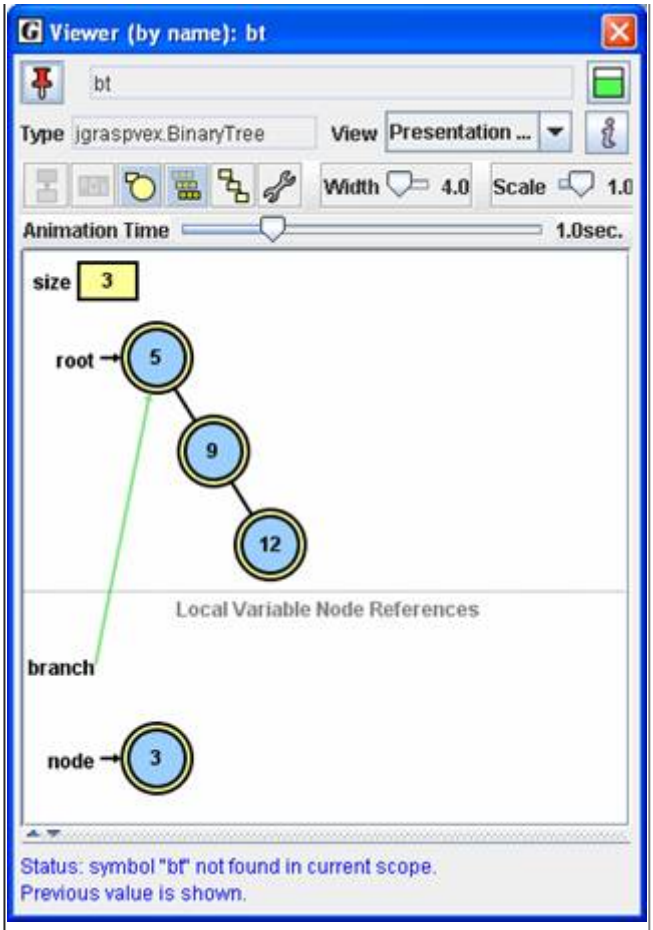

Figure 3. View after local node has been created and is about to be added to the binary tree

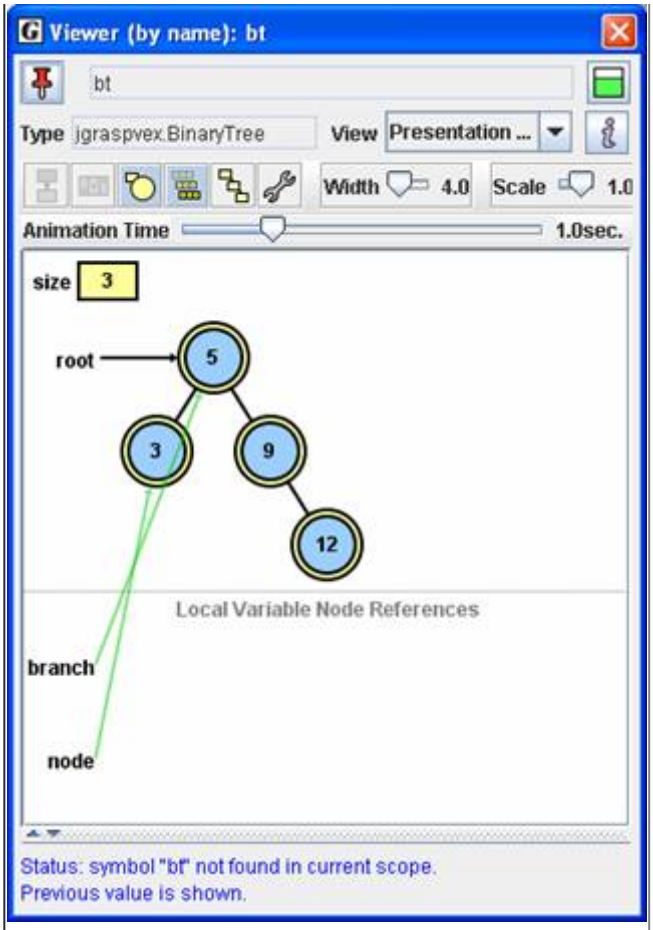

Figure 4. View after the node has "moved" from the local space into the binary tree and prior to size being updated

Results & contributions

Six controlled experiments were conducted to test various hypotheses. Experiments I and II were conducted using singly linked lists [Jain et al. 2006], Experiments III and IV were conducted using linked binary trees [Cross et al. 2007], Experiment V was conducted using min and max heaps, and finally Experiment VI was conducted using linked priority queues. Since for each experiment more than one response variable was measured, Hotelling's T^2 statistical method was used for multivariate data analysis.

The goal of Experiments I and III was to determine if students would be able to code more accurately and in less time using the jGRASP data structure viewers for a relatively easy (singly linked list) and a relatively hard (linked binary tree) to understand data structure. Students were asked to implement basic operations for each data structure. The group that performed the tasks using the jGRASP viewers performed significantly better than the other group which did not use the viewers. This means that students should be able to transition from conceptual knowledge to implementation easily for both relatively easy and hard to understand data structures that are taught in details during lectures.

The goal of Experiments II and IV was to determine if students would be able to find and correct more logical errors accurately and faster using jGRASP viewers for a relatively easy (singly linked list) and a relatively hard (linked binary

tree) to understand data structure. Students were provided code implementation with multiple logical errors. Their tasks consisted of locating and documenting the errors on paper, and then correcting the errors using jGRASP. It was observed that the group using viewers not only detected and corrected more errors in less time, but they also introduced fewer logical errors in the process. It was noticed that for Experiments I and II, there was not a statistically significant improved in the time taken to complete the tasks. In later experiments there is a clear improvement in the time taken by the group that uses viewer. A likely reason for the initial results is that students were inexperienced in pointer implementation.

The results of the on the paper-based surveys, indicated that students rated data structures covered abstractly in class as "difficult to understand" even though historically these are not that difficult. Experiment V was conducted using min-max heap to test if students would be able to transition from concept to implementation faster and more accurately using jGRASP viewers for data structures that are covered only conceptually in lectures. Students were given a min heap implementation, which they were asked to understand the code and convert into a max heap implementation and additionally implement other related operations. It was found that the group using viewers was able to complete the tasks more accurately and in less time. Thus viewers can be used outside of classroom to understand concepts and transition to implementation.

Experiment VI was conducted using linked priority queue to test if students would be able to apply concepts for data structures that were not covered in lectures faster and more accurately using jGRASP viewers. Students were provided with detailed conceptual explanation of the priority queue data structure. All the students were introduced to this data structure for the first time. It was found that the group using the jGRASP viewers was able to complete the tasks more accurately and in less time.

Finally, a questionnaire was conducted to evaluate the user interface aspects of the jGRASP debugger and the viewer window. It was found that students who knew how to use the debugger, only needed approximately took two to three minutes to learn to use the viewers. Minor interface redesign is currently in progress. Students also reported that stepping back during the debugging process would be very useful as that would allow different states of the data structure to be compared. Due to technical issues in Java 1.5, this feature will be considered after Java version 1.6 is released.

Initial comparison of average scores in exams and quizzes of students in the two experimental groups (i.e., the treatment group using viewers and control group that did not use viewers) in COMP2210 shows that the treatment group out performed the control group. A set of follow-on experiments to test if jGRASP viewers help with retention of concepts are recommended. It would also be useful to measure the amount of time spent in different activities while debugging (such as reading and editing source code and interacting with viewers) to determine of these for which activities jGRASP viewers are most helpful.

Data structure implementations from widely adopted CS2 textbooks are being tested for compatibility with the current jGRASP viewers, and the results so far are very positive. For the five textbooks tested, approximately 70% of data structures were recognized correctly by the structure identifier in jGRASP version 1.8.5 Beta 2. For the 30% that were not recognized the structure identifier could be manually configured to recognize and render the viewers. The main reason for lack of recognition was limited heuristics. As the heuristics become more comprehensive in the successive beta versions of jGRASP 1.8.5, the viewers should be able to automatically recognize over 95% of the textbook implementations for data structures in Java where the class name and fields are commonly used English identifiers.

References:

[Cross et al. 2007] CROSS, J.H., HENDRIX, D.T., JAIN, J., AND BAROWSKI, L.A. 2007. Dynamic Object Viewers for Data Structures. ACM SIGCSE, Technical Symposium on Computer Science Education, Kentucky, USA, March 7-10, 2007.

[Hamer 2004] HAMER, J. A lightweight visualizer for Java. Proceedings of Third Progam Visualization Workshop, July 1-2, 2004, 55-61.

[Hendrix et al. 2004] HENDRIX, D.T., CROSS, J.H., AND BAROWSKI, L.A. 2004. An extensible framework for providing dynamic data structure visualizations in a lightweight IDE. In Proceedings of the 35th SIGCSE technical symposium on Computer science education (SIGCSE) 2004, pp.387-391.

[Hendrix et. al 2006] HENDRIX, D.T., CROSS, J.H., JAIN, J., AND BAROWSKI, L.A. 2006 "Providing Data Structure Animations in a Lightweight IDE," 11th Annual Conference on Innovation and Technology in Computer Science Education, Bologna, Italy, June 26-28, 2006.

[Jain et al. 2005a] JAIN, J., CROSS, J., AND HENDRIX, D.T. 2005. Qualitative Comparison of Systems Facilitating Data Structure Visualization. Proceedings of the 43rd Southeast ACM Conference. Kennesaw, GA.

[Jain et al. 2005b] JAIN, J., BILLOR, N., HENDRIX, D.T., AND CROSS, J. H. 2005. Survey to Investigate Data Structure Understanding. Submitted to the International Conference on Statistics, Combinatorics, Mathematics and Applications, Auburn, AL, December 2-4, 2005.

[Jain et al. 2006] JAIN, J., CROSS, J., HENDRIX, D., AND BAROWSKI, L. 2006. Experimental Evaluation of Animated-Verifying Object Viewers for Java. ACM Symposium on Software Visualization (SoftVis), September 4-5, Brighton, UK, 2006.

[Kannusmaki et al. 2004] KANNUSMAKI, O., MORENO, A., MYLLER, N., SUTINEN, E. What a novice wants: students using program visualization in distance programming course. Proceedings of Third Progam Visualization Workshop, July 1-2, 2004, 126-133.

[Naps 2005] NAPS, T. JHAVÉ: supporting algorithm visualization. IEEE Computer Graphics and Applications, Sep-Oct 2005, 49-55.

[Zeller 2001] ZELLER, A. Visual Debugging with DDD. Dr. Dobb's, July 2001 (http://www.ddj.com/184404519).