Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 224 (2009) 141–149
www.elsevier.com/locate/entcs

Integrating Multiple Approaches for Interacting with Dynamic Data Structure Visualizations

James H. Cross II, T. Dean Hendrix, and Larry A. Barowski[1]

Department of Computer Science and Software Engineering
Auburn University
Auburn, Alabama 36849-5437 USA

Abstract

jGRASP 1.8.7 has integrated three approaches for interacting with its dynamic viewers for data structures: the debugger, the workbench, and a new text-based interactions tab that allows individual Java statements to be executed and expressions to be evaluated. While each of these approaches is distinct and can be used independently of the others, they can also be used together to provide a complementary set of interactions with the dynamic viewers. In order to integrate these approaches, the jGRASP visual debugger, workbench, and viewers had to be significantly redesigned. During this process, the structure identifier, which provides for the identification and rendering of common data structures, was also greatly improved by examining the examples from 20 data structure textbooks. The overall result of this integration effort is a highly flexible approach for user interaction with the dynamic data structure visualizations generated by a robust structure identifier.

Keywords: Program visualization, data structures, jGRASP, debugger, workbench, interactions.

1 Introduction

Visualizations of data structures have been used in limited ways for many years. To overcome a major obstacle to widespread use, namely lack of easy access and use, the jGRASP[2] IDE provides a structure identifier that attempts to identify and render traditional abstract visualizations for common data structures such as stacks, queues, linked lists, binary trees, heaps, and hash tables [2]. These are dynamic visualizations in that they are generated while running the users program in debug mode. This technique helps bridge the gap between implementation and the conceptual view of data structures, since the visualizations can be based on the users own code.

[1] Email: crossjh | hendrtd | barowla @auburn.edu
[2] The jGRASP research project is funded, in part, by a grant from the National Science Foundation.

doi:10.1016/j.entcs.2008.12.058

jGRASP 1.8.7 provides numerous ways for the user to interact with the data structure visualizations: (1) the debugger, (2) the workbench, and (3) a new text-based interactions tab. Each of these is briefly described below and then more fully with examples in the sections that follow.

The most common way to open a viewer on a data structure object is via the debugger. The user simply sets a breakpoint on a statement near the creation of the data structure, and runs the program in debug mode. After the program stops at the breakpoint, the user single steps as necessary until the object is created, and then opens a viewer on the object by dragging it from the debug tab. As the viewer is opened, the structure identifier determines the particular type of data structure and then renders it appropriately. When the user steps through the executing program, the viewer is updated to show the effects.

In addition to interacting with the data structure visualizations via the debugger, jGRASP allows the user to interact with viewers via the workbench. Objects can be created and their methods invoked via menus and buttons on the UML class diagram and/or the source code edit windows. When the user invokes a method on the object (e.g., to insert a node into the data structure), the visualization is updated to show the effect of the method.

The third approach for interacting with the viewers is a text-based interactions tab, which is essentially a Java interpreter. That is, when the user enters a Java source statement and presses the ENTER key, the statement is immediately executed. If an expression is entered, it is evaluated and its value is displayed. This means that while at a breakpoint, the user can interact with any of the variables in the debug tab or any variables on the workbench. In addition, if the user creates an instance of a class via the interactions tab (e.g., LinkedList list = new LinkedList();), list is shown on the workbench. A viewer can be opened in the usual way by dragging the reference from the debug tab or workbench. Once opened, the viewer is updated as appropriate when statements or expressions are entered in the interactions tab. In the following sections, we discuss related work, and then we provide an extended linked list example to illustrate the integration of the debugger, workbench, and text-based interactions with the viewers for data structures. This is followed by examples of other common data structures generated by jGRASP. A brief summary of the integration effort is presented, including future integration tasks, and then we close with concluding remarks.

2 Related Work

Both the method and degree of user interaction with software visualizations have been shown to be primary contributors to effectiveness. Research indicates that passive modes of interaction with visualizations, e.g., only watching an animation of an algorithms behavior, are not as effective as more active engagement strategies, e.g., having the user manipulate elements of the visualization or respond to prompts [12,7]. The context in which the visualization appears has also been shown to play a vital role in effectiveness [5]. These issues are now widely seen as fundamental to

the advancement of software visualization research in education [9,11].

We hold the pragmatic view that much of the learning and indeed much of the opportunity for learning occurs while a student is actively working on a source code implementation of an algorithm or data structure. Thus, we provide software visualizations that are directly coupled to real source code in the context of a full-featured IDE. This is similar to "scenario three" of user interaction described by Naps [11].

The three basic software visualization interaction techniques that jGRASP offers are based directly on common tools and idioms available in various IDEs: the debugger, object workbench, and interactions pane. The debugging process and the use of a debugger are central to almost all courses in which programming is involved and is supported by almost all IDEs. In addition, the use of a debugger as a pedagogical tool and as an important application area for software visualization has long been recognized [1,3]. The object workbench paradigm has been made popular by BlueJ (www.bluej.org), a popular IDE for early programming courses. In BlueJ, students use menus and dialogs to create object instances for the workbench and to invoke methods on those instances, without the need for a running program. In jGRASP, this notion of workbench has been extended to allow the user to open type-specific viewers on objects or primitives, as well as a viewer that identifies data structures on the fly. DrJava (www.drjava.org) has demonstrated the utility of an interactions tab for experimenting with Java source statements and expressions without the necessity of compiling and running a complete program. This is especially useful in a classroom setting. In jGRASP, we have extended the basic notion of the interactions tab in several ways. Most importantly, it is fully integrated with the debugger, workbench, and viewers.

The approach we have taken for data structure viewers in jGRASP is to automatically generate the visualization from the users executing source code and then to dynamically update it as the user steps through the source code in either debug mode or workbench mode, or as statements are executed in the interactions tab. This general approach is somewhat similar to the method used in Jeliot [6]. 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 [13]. The data structure visualization in DDD shows each object with its fields and shows field pointers and reference edges. 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 JHAVE [10]. We also decided against modifying the users source code as is required by systems such as LJV [4]. Our philosophy is that for visualizations to have the

most impact on program understanding they must be generated as needed from the users actual program during routine development and the user should be allowed to interact with the visualizations through commonly-used features of the IDE.

3 An Example – Integrated User Interaction

To see a meaningful visualization of a data structure in a viewer requires that an instance be created and its methods be invoked. The integrated approach in jGRASP allows this to be done by (1) using the debugger in the traditional way, (2) using the workbench menus and dialogs from the UML window or the edit window, and (3) entering source code into the interactions tab for direct execution. In each of these approaches, the viewer is opened on an instance by dragging it from the debug or workbench tab. Below, we will begin by using the debugger approach and then we will show how the user can interact with the visualization using both workbench and text-based approaches.

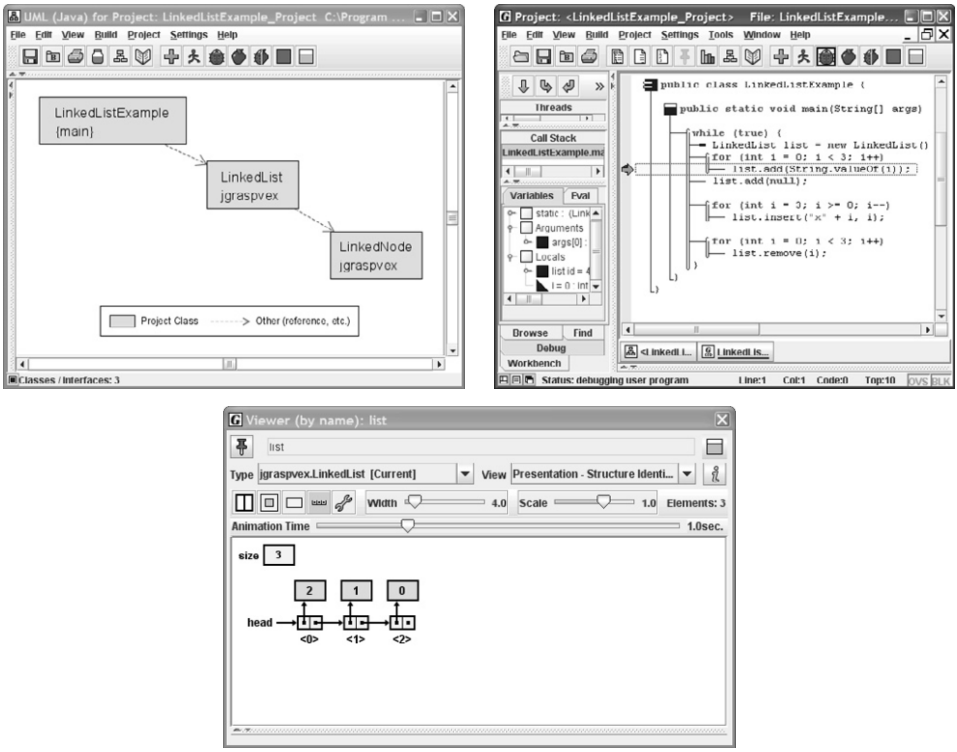

Fig. 1. (a) UML diagram for LinkedListExample (b) Method main at breakpoint (c) View of list after three elements added

3.1 Using the Debugger

Consider the program LinkedListExample.java, which is provided with the jGRASP distribution. The UML class diagram in Figure 1(a) shows the dependencies among LinkedListExample, LinkedList, and LinkedNode. The LinkedList class is intended

to be representative of a "textbook" example or of what a student or instructor may write. Figure 1(b) shows LinkedListExample.java stopped at a breakpoint where the list.add method is about to be invoked. Prior to stopping at the breakpoint, list was assigned to an instance of LinkedList, and thus it appears in the Variables tab of the Debug window. To open a viewer on list, the user simply drags it from the Debug window. During the process of opening the viewer, the Data Structure Identifier mechanism in jGRASP determines, in this case, that the object is a linked list structure and opens the appropriate viewer. As the user steps through the program, the nodes appear in the viewer. Figure 1(c) shows the visualization of list after three elements have been added.

The debugger approach is perhaps the most natural way for students to interact with the viewers. Since it involves stepping through a program, the visualization reinforces or clarifies the effect of each step by providing live feedback at a conceptual level. For example, Figure 2(a) shows the program after stepping into the insert method. The statement about to be executed will complete the linking of node into

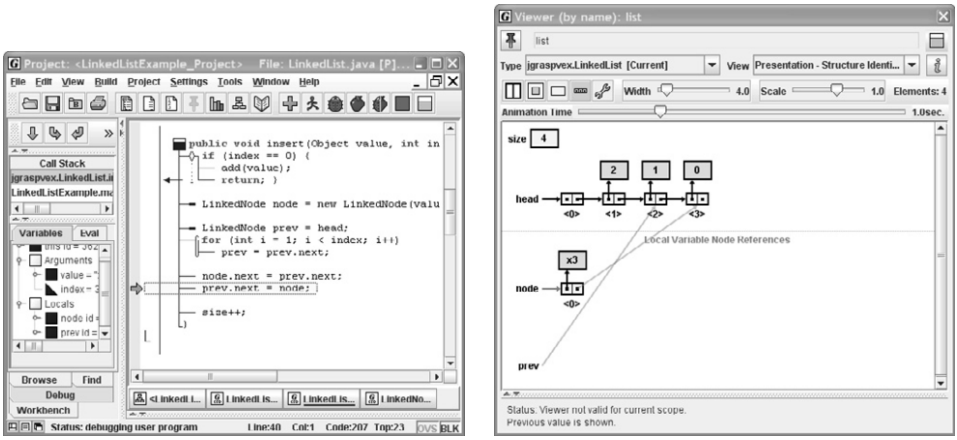

Fig. 2. (a) Stepping in list insert method (b) View of list showing details of insert

the data structure. Figure 2(b) shows the synchronized view of list at this point in the program. The visualization clearly indicates that prev.next is pointing to the last node in the structure (labeled "⟨3⟩"). As soon as the statement "prev.next = node;" is executed, the new node with value "x3" will move up into the structure in an animated fashion. The visualization in the viewer allows the student to make the connection between the implementation of the insert method and the concept of inserting an element in a linked list.

3.2 Using the Workbench

In the workbench approach for interacting with data structure visualizations, the user creates one or more instances of the class using menus or buttons on the UML class diagram or source code editing windows. From the UML diagram, this can be done by selecting Create New Instance on the right-click menu for the class, which pops up a Create instance dialog for class LinkedList as shown in Figure 3 below.

When the instance is created, it appears on Workbench tab, similar to the way

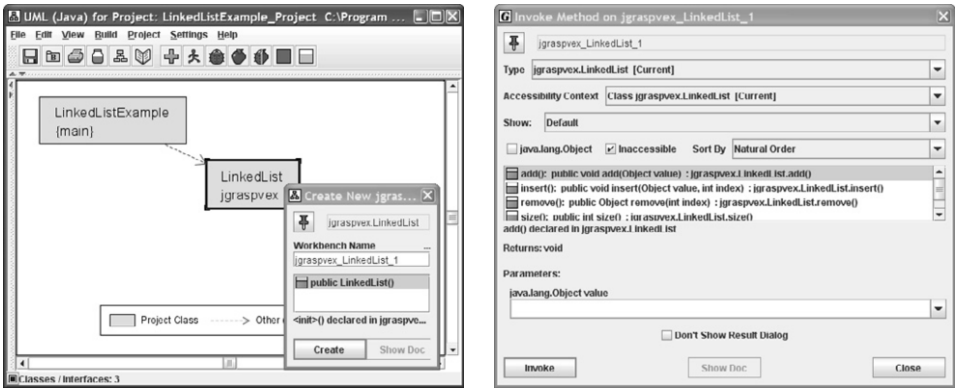

Fig. 3. (a) Creating instance from UML diagram (b) Invoke method dialog

list appeared in the Debug tab in Figure 1 above. As before, a viewer is opened on the instance by dragging it from the workbench. The Invoke Method dialog for the instance is popped up by either right-clicking on the instance and selecting Invoke Method or by clicking the Invoke Method button on the viewer (upper right corner). Figure 3 shows the Invoke Method dialog for jgraspvex-LikedList-1 with the add method selected. When a method is invoked, the viewer on the instance is updated to show the new state. Thus, the user can interact with the data structure visualizations in the viewer by invoking a sequence to add, insert, and/or remove methods. The operations just described could have been performed alone or in the context of a running program. In the latter case, the operations could also have been performed on an instance of LinkedList created by the program itself.

3.3 Using Text-Based Interactions

The new Interactions tab in jGRASP 1.8.7 provides a Java interpreter that allows the user to enter expressions and statements and have them immediately evaluated/executed. To use this feature in a stand-alone fashion with respect to data structure visualizations, the user could enter the code to create an instance and then enter statements that invoke methods on the instance. The advantage of this approach is that it allows the user to enter actual Java statements and execute them without having to enter and run an entire program, though the interactions tab can also be used to interact with elements in a running program. The integration with the workbench means that when a variable is declared and initialized (e.g., LinkedList myList = new LinkedList();), it appears on the workbench. This allows the user to open a viewer on the variable myList. Once the viewer is opened, any effects of the statements entered in the Interactions tab involving the methods invoked on myList will be seen in the viewer. To illustrate the degree of integration of the text-based interactions, consider our original example that we were running in debug mode. Figure 4(a) shows LinkedListExample stopped at the second for loop. Three list.insert method calls are shown in the Interactions tab. As these

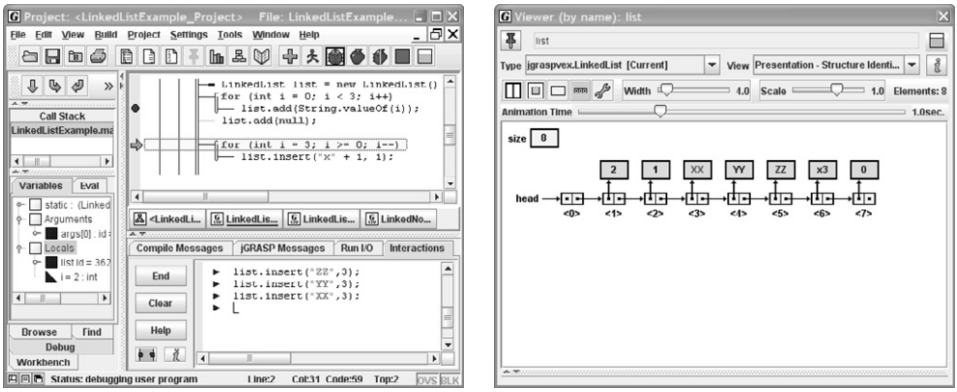

Fig. 4. (a) Interactions (b) Viewer for list after interactions

statements were entered, "ZZ", "YY", and "XX" respectively were inserted at location 3. As each node was entered, the existing nodes were moved to the right with the final result shown in Figure 4(b).

4 Examples of Other Data Structures

The jGRASP data structure identifier mechanism, which provides for the identification and rendering of common data structures, has been greatly improved by examining the examples from 20 data structure textbooks. The detailed results of the testing are described in [8]. Example visualizations now recognized by the mechanism are shown in Figure 5. These were generated from the example source code provided with jGRASP and include a queue, doubly linked list, binary tree, binary heap, red-black tree, and hash table.

5 Summary of Integration

The debugger, workbench, and interactions in jGRASP are flexible and well-integrated. The workbench and interactions may be used alone, together, or in conjunction with a program that is being debugged. Any object visible from the debugger, workbench, or in a viewer, including any sub-component object, can be placed on the workbench. Any object or primitive value visible from the debugger, workbench, or in a viewer, including any sub-component, can be displayed separately in a viewer. Objects created in interactions appear on the workbench. Non-void return values from workbench-initiated method invocations are displayed in viewers. Items on the workbench and in viewers are named, and those names may be referenced by source code in interactions, or in expressions passed as parameters to workbench object creation and method invocation dialogs. Java expressions (not just objects) can be displayed in a viewer by dragging them from the "Eval" tab in the debugger or by using an interface that directly launches a viewer for an expression.

More integration is planned for the future. We intend to make it possible to echo

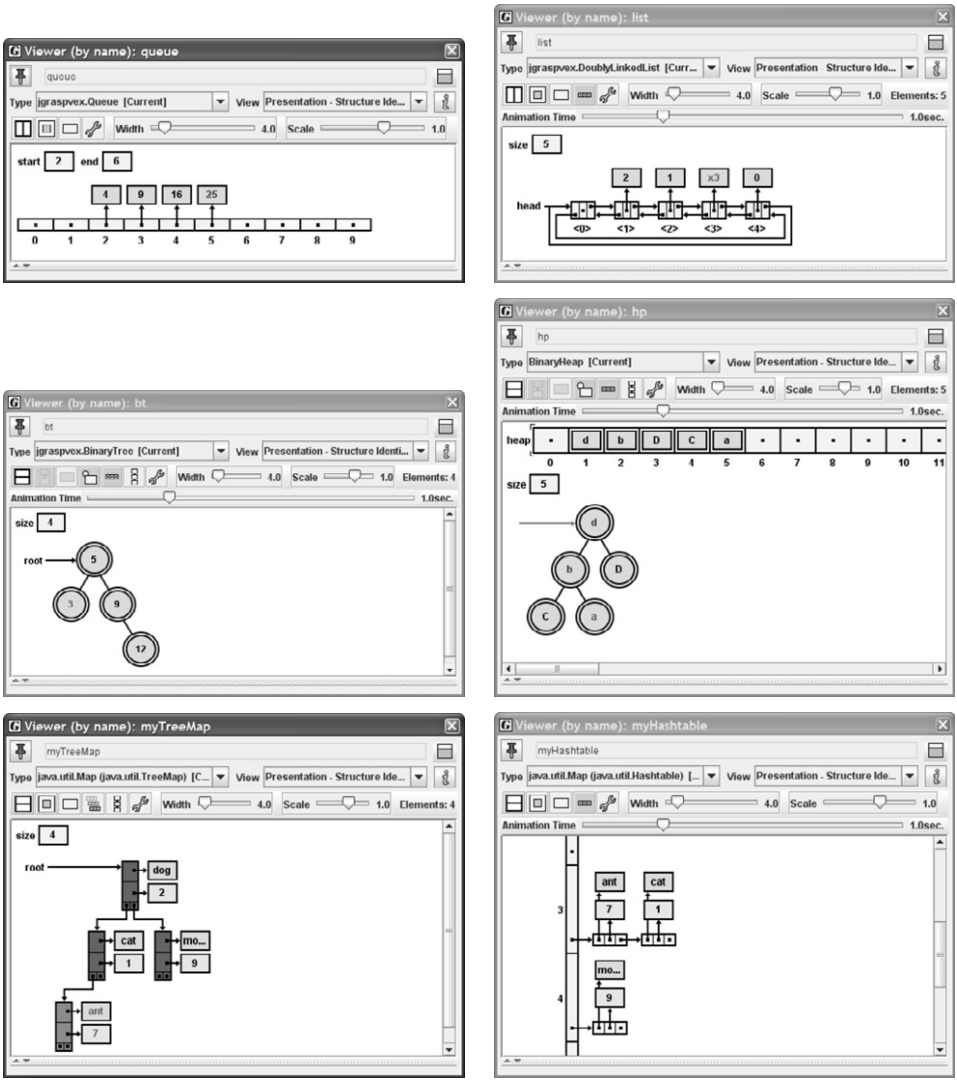

Fig. 5. Example visualizations automatically generated by jGRASP

workbench object creation and method invocation actions to the interactions tab as source code text. Also, special syntax will be recognized by interactions so that viewers for values and expressions can be launched directly from the interactions tab without leaving the text-based interface.

6 Conclusion

The data structure visualizations provided by jGRASP are intended to support teaching and learning activities in courses that include data structures. In U.S. computer science programs, this would typically be CS2 or CS3. The highly visual debugger in jGRASP, which provides a natural interface for the data structure visualizations, has also been a major strength for CS1 students as they learn the

basics of object-oriented programming. Those students who use BlueJ in their CS1 course are likely to be comfortable with the workbench approach for developing their programs, and those who use DrJava are familiar with the text-based interactions approach. By integrating these three approaches, students in CS2 and CS3 are allowed to interact with their data structure visualizations in the manner with which they are most comfortable. Furthermore, the integration allows the students to mix and match the operational aspects of each approach in a seamless manner.

The process of integrating the approaches required that the jGRASP visual debugger, workbench, and viewers be significantly redesigned. The redesign of the viewers included significant improvements to the data structure identifier mechanism which generates the visualizations. Example programs from 20 data structure textbooks were used to tune the data structure identifier mechanism. The overall effect of the redesign should be a highly flexible approach for user interaction with a rich set of automatically generated dynamic data structure visualizations.

References

[1] Baecker, R., C. DiGiano and A. Marcus, *Software visualization for debugging*, Communications of the ACM **40** (1997), pp. 44–54.

[2] Cross, J. H., T. D.Hendrix, J. Jain and L. A. Barowski, *Dynamic object viewers for data structures*, in: *Proceedings of the SIGCSE 2007 Technical Symposium*, 2007, pp. 4–8.

[3] Cross, J. H., T. D. Hendrix and L. A. Barowski, *Using the debugger as an integral part of teaching cs 1*, in: *Proceedings of Frontiers in Education 2002*, 2002, pp. 55–58.

[4] Hamer, J., *A lightweight visualizer for java*, in: *Proceedings of Third Progam Visualization Workshop*, 2004, pp. 55–61.

[5] Hansen, S. R., N. H. Narayanan and M. Hegarty, *Designing educationally effective algorithm visualizations: Embedding analogies and animations in hypermedia*, Journal of Visual Languages and Computing **13** (2002), pp. 291–317.

[6] Kannusmaki, O., A. Moreno, N. Myller and E. Sutinen, *What a novice wants: students using program visualization in distance programming course*, in: *Proceedings of Third Progam Visualization Workshop*, 2004, pp. 126–133.

[7] Lauer, T., *Learner interaction with algorithm visualizations: Viewing vs. changing vs. constructing*, in: *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE 2006)*, 2006, pp. 202–206.

[8] Montgomery, L. N., J. H. Cross, T. D. Hendrix and L. A. Barowski, *Testing the jgrasp structure identifier with data structure examples from textbooks*, in: *Proceedings of the 46th ACM Southeast Conference*, 2008, pp. 198–203.

[9] Naps, T., *Instructional interaction with algorithm visualizations*, Journal of Computing Sciences in Colleges **16** (2000), pp. 7–8.

[10] Naps, T., *Jhave: supporting algorithm visualization*, IEEE Computer Graphics and Applications **Sep/Oct** (2005), pp. 49–55.

[11] Naps, T. and G. Roessling, *Development of xml-based tools to support user interaction with algorithm visualization*, ACM SIGCSE Bulletin **37** (2005), pp. 123–138.

[12] Stasko, J. and A. Lawrence, *Empirically assessing algorithm animations as learning aids*, in: *Software Visualization: Programming as a Multimedia Experience*, 1998, pp. 419–438.

[13] Zeller, A., *Visual debugging with ddd*, Dr. Dobb's Journal **July** (2001).