

Visualizing Programs with Jeliot 3

Andrés Moreno, Niko Myller, Erkki Sutinen
Department of Computer Science
University of Joensuu
P.O. Box 111
FIN-80101 Joensuu, Finland
+358-13-251 7928
{amoreno, nmyller, sutinen}@cs.joensuu.fi

Mordechai Ben-Ari
Department of Science Teaching
Weizmann Institute of Science
Rehovot 76100, Israel
+972-8-934 2940
moti.ben-ari@weizmann.ac.il

ABSTRACT

We present a program visualization tool called Jeliot 3 that is designed to aid novice students to learn procedural and object oriented programming. The key feature of Jeliot is the fully or semi-automatic visualization of the data and control flows. The development process of Jeliot has been research-oriented, meaning that all the different versions have had their own research agenda rising from the design of the previous version and their empirical evaluations. In this process, the user interface and visualization has evolved to better suit the targeted audience, which in the case of Jeliot 3, is novice programmers. In this paper we explain the model for the system and introduce the features of the user interface and visualization engine. Moreover, we have developed an intermediate language that is used to decouple the interpretation of the program from its visualization. This has led to a modular design that permits both internal and external extensibility.

Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education - *Computer Science education*; H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems - *Animations*; I.6.8 [Simulation and Modeling]: Types of Simulation - *Animation, Visual*

General Terms

Human Factors.

Keywords

Program visualization, novice programming.

1. INTRODUCTION

When considering the possibility of visualizing an algorithm or a program, it may appear that visualization is a superior way to illustrate their behavior. In particular, when students are learning algorithms or programming, this kind of tool seems to be an excellent learning resource. However, it has been proved during several empirical experiments that an animation of the running

algorithm only helps students to learn if it somehow cognitively engages them and is specially targeted for the particular user population (e.g. for novices) [4, 11].

The results achieved have driven researchers to reformulate their research questions. It has become clear that the media itself does not have a strong effect on the learning outcomes. The effect of visualization is in the organization of the content, in the way the subject is taught (i.e. are the students passive observers or active learners), and how it suits the particular learner population. This is also in line with more general research done by educational psychologists on multimedia learning [8].

In Jeliot, the idea is to involve the students in the construction of their own programs and at the same time examine a visual representation of the programs' execution. During this process they acquire a mental model of the computation that helps them to understand the constructs of programming. Furthermore, the model can be used to acquire new knowledge and the vocabulary used to discuss programs and programming concepts. Thus the students are engaged with the tool and are learning by doing.

Object-oriented programming is getting much attention and object-oriented languages such as Java are used as the first language to teach programming. The visualization of object oriented concepts such as objects and inheritance are important, because these concepts are not easily grasped by novice programmers. This was one of the issues that we wanted to concentrate on in Jeliot 3 [9].

2. PREVIOUS WORK

BlueJ [1] is one of the first systems developed to teach introductory object oriented programming. The key feature of the system is the static visualization of the class structure as a UML diagram. Furthermore, it allows the learner to interact with the objects by creating them, calling their methods and inspecting their state with easy-to-use menus and dialogs. However, it does not provide any dynamic visualization of the program, which is the purpose of our system. Moreover, it is also possible to introduce a similar kind of visualization into Jeliot 3 with slight modifications.

Javavis [10] is a system developed from the same idea of using the Java Debugging Interface (JDI) to obtain information about the runtime behavior of the program. It visualizes the state of the program and its changes during execution. The system is not meant for novices, because the visualization it produces assumes that students are familiar with UML and the basics of programming. However, this kind of system could be very useful for advanced courses in programming.

© ACM, 2004. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in the Proceedings of the Advanced Visual Interfaces (AVI 2004) Conference. <http://doi.acm.org/10.1145/nnnnnn.nnnnnn>

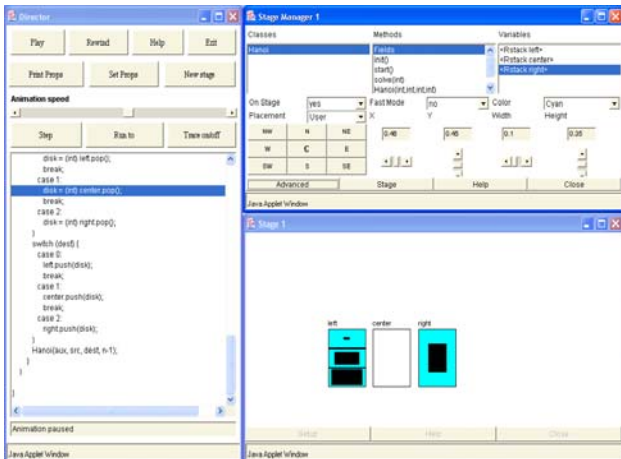


Figure 1: User interfaces of Jeliot I.

3. HISTORY OF JELIOT

The development of the Jeliot family [2, 6] started almost ten years ago when the first system *Eliot* [5] was developed to help in the production of algorithm animations. After *Eliot* two other systems have been developed: *Jeliot I* [13] and *Jeliot 2000* [4].

The development process of Jeliot has been research-oriented, meaning that all the versions have had their own research agenda rising from the previous versions' design and empirical evaluations. All these systems have been implemented in different environments and a new version has been developed either to extend the possibilities for visualization or to support different user populations. The first versions, *Eliot* and *Jeliot I*, shared the main goal, which was to ease the production of algorithms animations. The *Jeliot I* implementation allowed it to be used on the Internet, making *Jeliot's* use distance independent. *Jeliot 2000* was especially designed for novice learners, whereas *Jeliot 3* [9] is a generalization of the work done with *Jeliot 2000*; extending it to visualize object oriented concepts.

During the development and evaluation cycle of Jeliot, it has been learned that there is no one best formula for all learning needs, but there should be several items in the learning environment from which the learner can select the ones she needs [2]. This means that we should give students the possibility to use different kinds of visualizations with various orientations leading to a stage where an extendable and modular system is needed as a basis for this development.

3.1 Evolution of the User Interface

The user interfaces of *Eliot* and *Jeliot I* consisted of multiple windows (see Figure 1). *Eliot* used the resources provided by X-Windows system, whereas *Jeliot I* relied on the Java support of internet browsers. A common animation scenario was displayed on at least three different windows, which could be placed anywhere on the screen, with the control buttons divided among the windows. Moreover, users had to place the visualization objects by means of dialogs that provided great level of detail. However, setting all these details could hinder the understanding of the resulting animation, because user would focus on these details not on the animation itself. These factors made the usage of the systems more challenging for novices. In an empirical evaluation, novices found *Jeliot I* too complex to use [7].

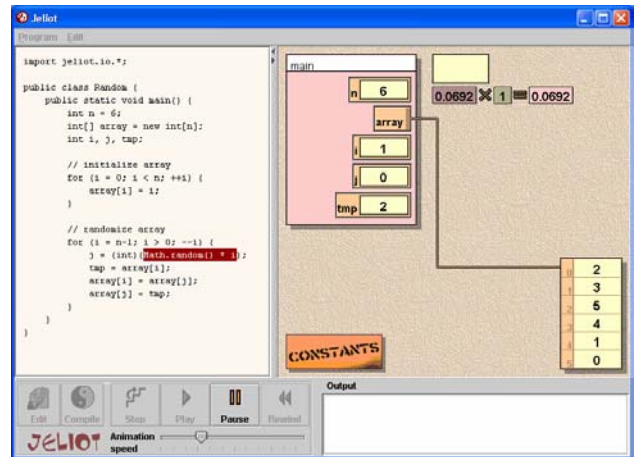


Figure 2: User interface of Jeliot 2000.

The user interface of *Jeliot 2000* is illustrated in Figure 2. It consists of just one window with several panes. The automatic visualization of *Jeliot 2000* also implies a simplified user interface. There is no need to set up any parameters before starting the animation of a program. To start the animation just two buttons ("Compile" and "Play") are needed. The buttons used to control the animation resemble a VCR control panel, making it easy for novices to identify their meaning. Finally *Jeliot 2000* is a single application, overcoming the difficulties that the server-client model of *Jeliot I* presented [7].

3.2 Evolution of the Visualization

Eliot and *Jeliot I* concentrated their visualization on data visualization (i.e. they visualized only variables); the animation consisted of values moving from one variable to another or comparisons with the other values. However the information provided by *Eliot* and *Jeliot I* animations was not descriptive enough for students. Lattu et al. [7] found that the visualization of the data flow was insufficient for the novice students, who also require visualization of the control flow and object structures. Moreover, novices require that even expression evaluation is fully visualized. Thus, control flow visualization and expression evaluation was added into *Jeliot 2000*. However, the shift in the visualization focus meant losing a level of abstraction in the visualization, while allowing a more complete visualization of the data and control flow of the program.

4. JELIOT 3

4.1 Goals

The goals of *Jeliot 3* were defined by previous experience with the *Jeliot* family in empirical evaluations. While maintaining the same basic features, new features were added to better suit the user population. The main goals of the systems are listed below:

- The system must be easy to use.
- The visualizations produced by the system should be consistent with the visualization in all cases.
- The visualizations produced by the system should be complete and continuous.
- The system should support the visualization of as large a subset of programs written in Java language as possible.
- The system should be extensible internally and externally.

The first three goals come from the fact that Jeliot 3 is intended for novice users, and in research on Jeliot [3, 7] and visual displays [11] these features have been found important for them.

Even though the system is intended for novices, it should support the visualization of as large a subset of programs written in Java language as possible. We also wanted to support the possibility to stay with Jeliot longer than the first couple of weeks. Jeliot 3 introduces object-oriented concepts, visualizing objects and inheritance, concepts which can be now discussed during the first courses on programming in an objects-first approach [1].

Extensibility has been a problem of the previous versions of Jeliot. Adding new features to the visualization has not been easy, thus with this new version, extensibility was one of the important issues. As stated before, students should have the possibility to use various visualizations in different stages of learning. Production of several visualizations requires that the system be extensible.

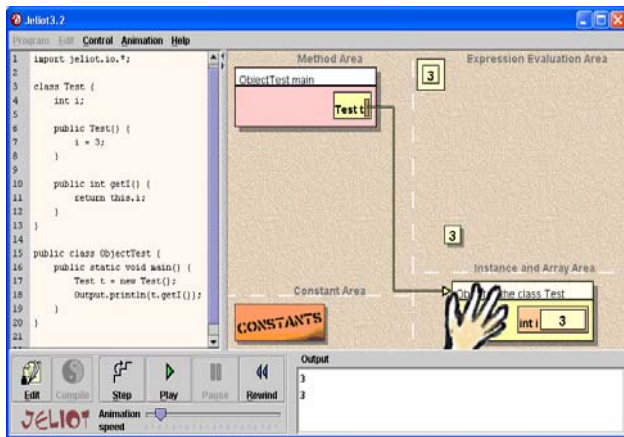


Figure 3: User interface of Jeliot 3.

4.2 User Interface

The design of the user interface is a crucial aspect in the tool for novice computer user. We used the user interface design from Jeliot 2000 as it was found usable and simple for novices [3]. We have added extra menus and shortcut keys to make the usage of the software smoother. Line numbering was added to the code editor and view to make the referencing to the code easier.

The user interface of Jeliot 3 is illustrated in Figure 3. The menu bar consists of buttons and menus that can be used during the editing or visualization of the program. During visualization, the buttons are taken away to leave more space for the source code visualization in the code viewer. The control panel contains the VCR-like buttons to control the visualization in the visualization frame. If an error occurs during the execution of the program, an error viewer shows the reason and the code view highlights the area where the error possibly happened. If the program requests any input it is shown in the visualization frame. If any output is printed it will be shown in the output console at the bottom of the window.

4.3 Visualization

In the visualization of the programs we had several principles that we used to justify our decisions. We wanted to be as consistent as

possible to reduce the cognitive load of the student. As stated by Petre and Green [11], seeing the secondary notation of a graphical display, in our case the layout of the animation frame, is an acquired skill and novices do not have it. This means that the secondary notation has to be done as explicitly and consistently as possible. All the visualized components have their own area on the screen as shown in Figure 4 and they always appear in that area. Furthermore, the visualizations are formed as close as possible to the Java Language Specification as far as it has been pedagogically reasonable.

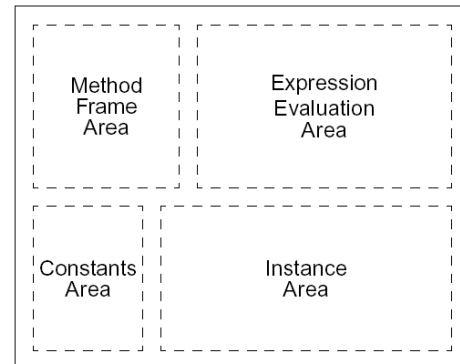


Figure 4: The structure of the animation frame in Jeliot 3.

We also used the results about multimedia learning from Mayer [8] as general guidelines. All the visualized material was coherent and complete in a sense that none of the visualized elements appear from nowhere, but each of the elements has its own place to appear. In addition, all the expressions and their subexpressions are evaluated and all the values shown so that student needs not guess where each value is coming from. Furthermore, the visualization and program code is linked with the code highlighting so that cause and effect could be identified.

All the explanations and the related expressions are always shown as close to each other as possible to help linking the value of the expression and the resulting explanation.

For object-oriented programming visualization, we have tried to use UML-like notation. The objects are shown as boxes that contain attributes and their values. The references are shown as lines connecting the object with the corresponding variable, allowing the object have several references at any moment.

4.4 Design and Implementation

The new program visualization system is based on the previously developed system, Jeliot 2000, and the research done with it [3]. However, designing a new system required a different approach to avoid the problems found in Jeliot 2000. In the design of the new version we wanted to reuse as much existing code as possible. Thus, we tried to design modular system that could use other programs as its components.

The visualization engine of Jeliot 2000 was easily reusable and found to be effective [3]. It was taken as the visualization engine of the new version. However, the Java interpreter of Jeliot 2000 was hand-crafted and the further development of it would have been difficult. We decided to use an already available open source Java interpreter, *DynamicJava*. It is almost fully Java compliant and it is written in Java. Thus, it had all the features that we were

looking for, full support for Java language and the possibility to integrate it easily into our system.

Two approaches were tried when designing the communication model between these two systems. The first approach was to form a description of the programs execution in XML, as proposed by Stratton [12]. However, we decided not to use the XML-based language because it would have made the program much heavier, and the well-formedness requirement of XML would have caused problems, especially with programs requesting input from the user. In the second approach, we designed an intermediate language between these two systems. This intermediate language consists of simple ASCII text lines that carry all the information needed to visualize the interpretation of a program. This means that our intermediate language is a result of the *interpretation* of the program by DynamicJava, not just another form to describe the source code.

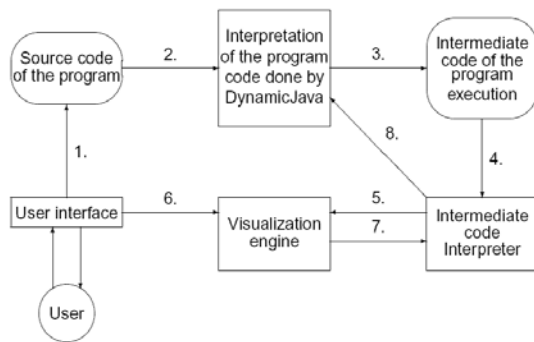


Figure 5: The functional structure of Jeliot 3

The functional structure of the Jeliot 3 is shown in the Figure 5. A user interacts with the user interface and creates the source code of the program (1). The source code is sent to the Java interpreter and the intermediate code is extracted (2 and 3). The intermediate code is interpreted and directions are given to the visualization engine (4 and 5). The user can control the animation by playing, pausing, rewinding or playing step-by-step the animation (6). Furthermore, the user can input data, for example, an integer or a string, to the program executed by the interpreter (6, 7 and 8).

The intermediate language provides a source of interpretation information that can be used for different visualizations. A new intermediate code interpreter and a visualization engine can be developed to produce different visualization of the same program (e.g. call tree visualization). Thus, Jeliot 3 can be extended internally with multiple visualizations of the same program. Secondly, the intermediate language can be interpreted again without Java interpreter, so the visualization can be stored in the form of the intermediate language and it will produce exactly the same visualization.

5. CONCLUSION

Jeliot 3 has improved previous versions of the Jeliot family by incorporating several new features: objects support, larger subset of programs accepted, improved error information, improved design, better extensibility, etc.

Jeliot 3 is ready to be used by teachers at introductory programming courses. Jeliot 3 can help in the early stages of these courses by providing clear semantics and by engaging students into the learning process. With Jeliot 3 visualizations, teachers

and students can share a graphical and verbal vocabulary that eases the discussion of programming concepts.

Jeliot 3 aspires also to be the base of future developments. Maintaining it stable and documented will encourage developers to create new visualization models. Those new features will adapt Jeliot 3 to different scenarios and could make it a more valuable tool. The open development of Jeliot 3 will stand in the basis of public licensing (GPL) and it will be coordinated from the University of Joensuu.

First evaluation of Jeliot 3 will be carried out in the beginning of the year 2004. Results of this evaluation will show us how to proceed in following versions.

6. REFERENCES

- [1] D. J. Barnes and M. Kölling. *Objects First with Java – A Practical Introduction using BlueJ*. Prentice Hall/Pearson Education, Reading, Massachusetts, USA, 2003.
- [2] M. Ben-Ari, N. Myller, E. Sutinen, and J. Tarhio. Perspectives on Program Animation with Jeliot. In S. Diehl, editor, *Software Visualization*, vol. 2269 of *Lecture Notes in Computer Science*, pages 31–45. Springer-Verlag, 2002.
- [3] R. Ben-Bassat Levy, M. Ben-Ari, and P. A. Uronen. The Jeliot 2000 program animation system. *Computers & Education*, 40(1):15–21, 2003.
- [4] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko. A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages & Computing*, 13(3):259–290, 2002.
- [5] S.-P. Lahtinen, E. Sutinen, and J. Tarhio. Automated Animation of Algorithms with Eliot. *Journal of Visual Languages and Computing*, 9(3):337–349, 1998.
- [6] Jeliot, WWW-page, <http://cs.joensuu.fi/jeliot>, 2003
- [7] M. Lattu, V. Meisalo, and J. Tarhio. A visualization tool as a demonstration aid. *Computers & Education*, 41(2):133–148, 2003.
- [8] R. E. Mayer. *Multimedia Learning*. Cambridge University Press, Cambridge, UK, 2001.
- [9] Andrés Moreno, Niko Myller. Producing an Educationally Effective and Usable Tool for Learning, the Case of the Jeliot Family. To appear in the *Proceedings of International Conference on Networked e-learning for European Universities*, Granada, Spain, 2003
- [10] R. Oechsle and T. Schmitt. JAVAVIS: Automatic Program Visualization with Object and Sequence Diagrams Using the Java Debug Interface (JDI). In S. Diehl, editor, *Software Visualization*, volume 2269 of *Lecture Notes in Computer Science*, pages 176–190. Springer-Verlag, 2002.
- [11] M. Petre. Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming. *Communication of the ACM*, 38(6):55–70, 1995.
- [12] D. Stratton. A Program Visualisation Meta-Language Proposal. In C. H. Lee, editor, *Proceedings of the 9th International Conference on Computers in Education /SchoolNet2001*, pages 601–609, Seoul, S. Korea, 2001.
- [13] E. Sutinen, J. Tarhio, and T. Teräsvirta. Easy Algorithm Animation on the Web. *Multimedia Tools and Applications*, 19(2):179–184, 2000