Informatics in Education, 2006, Vol. 5, No. 2, 195–206
© 2006 *Institute of Mathematics and Informatics, Vilnius*

Various Utilizations of an Open-Source Program Visualization Tool, Jeliot 3

Roman BEDNARIK, Andrés MORENO, Niko MYLLER
Department of Computer Science, University of Joensuu
PO Box 111, FI-80101, Finland
e-mail: firstname.surname@cs.joensuu.fi

Received: January 2006

Abstract. In this paper, we present an open-source program visualization tool, Jeliot 3. We discuss the design principles and philosophy that gave rise to this successful e-learning tool and to several other related environments. Beside Jeliot 3, we introduce three different environments, BlueJ, EJE, and JeCo that use Jeliot 3 as a plug-in to allow visualization of the program code. Another system, FADA, is a tool that was derived from Jeliot 3 but serves for different pedagogical goals. A community of users and developers of these projects has been created and supported, that allows for global and iterative improvements of the Jeliot 3 tool. This way, both academic research and feedback from the user community contribute to the development. We compare the presented approach of the tool development to some of the current tools and we discuss several instances evidencing a particular success.

Key words: e-learning, program visualization, open source development, GPL.

1. Introduction

Programming is a skill that, in the present society, becomes often necessary and required in everyday situations: actions such as modifying a piece of text or writing a macro in office software or setting the multimedia system to record a selected television program are just three examples of end-user programming. At the same time, programming is a complex task with multiple interrelated components, tradeoff decisions, and performance requirements that concern the whole process (Detienne, 2002; Hoc *et al.*, 1990). As well as with other skills, to become an expert in programming requires a deliberate practice (Ericsson, 2003). Central to programming is the ability to comprehend a computer program, so to establish a valid mental representation of the problem solved by the program. Because of the lack of knowledge and experience, novice programmers have problems with constructing the viable models of problems. Therefore, tools to support the construction of such models are desirable.

One of the approaches to support novice programmers in their comprehension task is algorithm and program visualization. Typically, a novice-oriented program visualization tool is used to teach programming concepts, for example, by showing some parts of a program graphically. This can mean for example, a class diagram or animation of the programs execution behavior are shown to the students. A number of visualization tools have been developed in previous years; however, the practical results have been inconclusive (for an analysis, see Hundhausen *et al.*, 2002). It has been also pointed out, that these tools are often designed in a uniform fashion (Bednarik *et al.*, 2005), without a contextual sensitivity to task and to learner's cognitive growth (e.g., Ben-Ari, 2001).

From a software engineering point of view, another reason behind the partial failure of program visualization tools can be the low number of iterations in the development life-cycle of the tool and the restrictions in the distribution policies. Particularly, usability and empirical evaluations of the tools are missing, the tools are not published freely, and the interaction between designer- and user communities is limited to an email support. Typically, a tool and the visualizations are designed by experts in programming and then tested in a laboratory setting or in a programming course. At the same time, the source-codes are not distributed along with the tool, the tool does not provide means for easy extensibility, and thus the community around the tool is restricted. We present another philosophy to the development of e-learning tools. Our intention is to 1) provide full access to the source codes, 2) allow for extensibility, modularity and integration with third-party environments, 3) evaluate regularly the outcomes of the tools, 4) and create and support user community.

In this paper, we introduce Jeliot 3 (Moreno *et al.*, 2004a) (see also `http://cs.joensuu.fi/jeliot`), a program visualization tool which is especially designed for novices. It has a simple user interface and the visualizations are complete, explaining the whole program execution to the student. Jeliot 3 can be used both to teach and to learn programming. Jeliot 3 is an open source program visualization tool that has been used to teach programming to novices for several years (Ben-Ari *et al.*, 2002, Ben-Bassat Levy *et al.*, 2003). We discuss the previous, current and future use cases and prospects of this e-learning tool.

The paper is organized as follows: we first introduce the concept of open source, and then we review a state-of-art of program visualization tools with respect to the openness and extensibility. Later we introduce Jeliot 3 and the family of its predecessors. The design philosophy introduced in the following section, the most interesting extensions and derivations are reviewed, demonstrating the success of the environment. Future plans are discussed and conclusions presented.

2. Open Source Approach

The open source approach to development promotes that applications are distributed together with their source codes. Developers, both internal and external, can inspect the source code of the software for security flaws, or they can modify it to better suit their needs. Different distribution licenses for open source software exist in order to specify what can be done with the available source code. The Berkeley Software Distribution (BSD) license and the General Public License (GPL) are the two of the most commonly used open licenses. The main difference between them is that software distributed under

BSD can be adopted into commercial software without restrictions. The GPL requires all subsequent software derived from GPL licensed software to be GPL licensed as well.

A great deal of current public applications is published under an open source license. Successful examples of GPL licensed software are the GNU/Linux operating system, or the Mozilla Foundation Firefox web browser. In the field of education, the Moodle, an online learning environment, is being widely adopted by educational institutions. The fact that these tools are distributed for free can partly explain their success and an increasing share on the market. However, the possibility to contribute to the development of these tools without restrictions has positively affected their success as well. All of them provide the means to let users adapt and extend the capabilities of the software beyond the original goal.

3. Overview of Program Visualization Tools

Many program visualization tools and environments have been developed to support the teaching and learning of programming. A complete overview of all the tools goes beyond the limitations of this paper[1]; however, Table 1 summarizes a representative set of tools for program/algorithm visualization, developed recently or being developed in academic research. Focusing on the aspects important to the current interest, it can be observed that

Table 1

An overview of present program/algorithm visualization tools with regard to their openness. The columns in order are: system name, implementation language, operating system on which the system can be run, is the system downloadable, is documentation available, licensing, does system's website contain discussion board or mailing list, has the system been update lately or is the project alive, is the system extensible, reference to the publication explaining the system

System	**Impl. Lang.**	**OS**	**DL**	**Doc.**	**Lic.**	**Disc. board/ mail. list**	**Upd./ proj. alive**	**Ext.**	**Ref.**
Animal	Java	Win/ Unix	Yes	Yes	Other[2]	No	Yes	Yes	Rössling & Freisleben, 2002
BlueJ	Java	Win/ Unix	Yes	Yes	Other	Yes	Yes	Yes	Kölling *et al.*, 2003
Dynalab	Pascal	Win/ Unix	Yes	No	Other	No	No	No	Boroni *et al.*, 1996
Jeliot	Java	Win/ Unix	Yes	Yes	GPL	Yes	Yes	Yes	Moreno *et al.*, 2004a
JIVE	Java	Win/ Unix	Yes	No	N/A	No	Yes	No	Gestwicki & Jayaraman, 2005
Matrix	Java	Win/ Unix	Yes	Yes	GPL	No	Yes	Yes	Korhonen *et al.*, 2004
PlanAni	Tcl/Tk	Win/ Unix	Yes	Yes	N/A	No	Yes	Yes	Sajaniemi & Kuittinen, 2003
Tango	C	Unix	Yes	Yes	Other	No	No	No	Stasko, 1990

[1]For a recent overview c.f. Diehl, S. (Ed.), Software Visualization. Vol. 2269 of Lecture Notes in Computer Science. Springer-Verlag, 2002.

[2]Other: usually the tools are free to download and distribute, however the source-codes are not available at the moment.

while most of the systems can be run on any of the current operating systems, only few systems are distributed freely under an open-source license, and even fewer provide a discussion area for the user community. In our view, we regard a system to be extensible if it supports either internal or external extensions with an application program interface (API) or other well defined interface.

4. Jeliot Family

The development of the Jeliot family (Ben-Ari *et al.*, 2002) began almost ten years ago when the first system Eliot (Lahtinen *et al.*, 1998) was developed to help in the production of algorithm animations. After Eliot, two other systems have been developed, namely Jeliot I (Sutinen *et al.*, 2003) (see also `http://cs.joensuu.fi/jeliot/jeliot.html`) and Jeliot 2000 (Ben-Bassat Levy *et al.*, 2003).

The development process of Jeliot has been research-oriented, meaning that all versions have had their own research agenda rising from the previous versions' design and empirical evaluations. The 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 (Moreno *et al.*, 2004a) is a generalization of the work done with Jeliot 2000; extending it to visualize object oriented concepts. Fig. 1 shows the interface of Jeliot 3 with an ongoing animation.

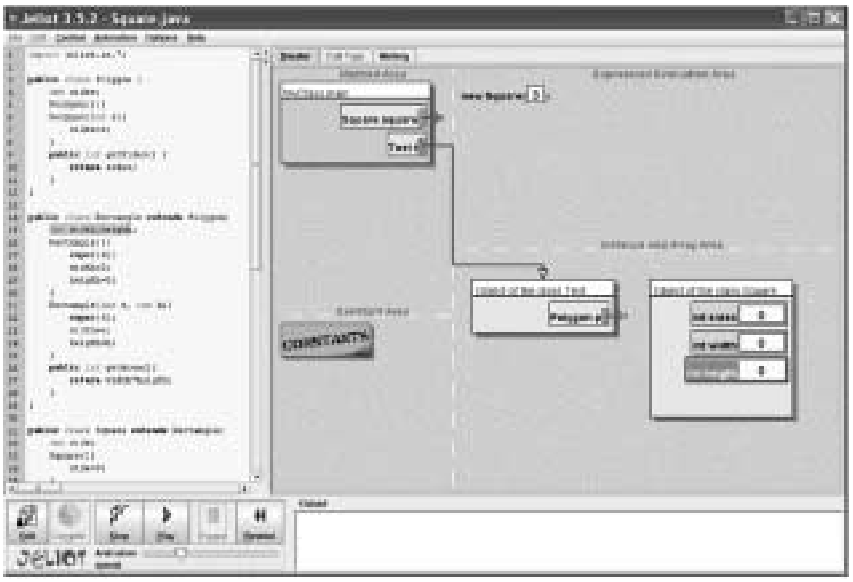


Fig. 1. Interface of Jeliot 3.

During the development and evaluation cycles of Jeliot, it has been learned that there is no one best formula for all learning needs, but there should be several features in the learning environment from which the learners can select the ones they need (Ben-Ari *et al.*, 2002). 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. This was our aim when we designed the Jeliot 3 system.

To support the extensions of Jeliot 3 we have published the source codes of the system under the GPL license. Furthermore, we have developed a website (see `http://cs.joensuu.fi/jeliot`) that contains materials for adopting as well as for modifying Jeliot 3. There are documentation, discussion forum, published research papers, and a mailing list, all available to support the exchange of information.

5. Extensibility and Reusability of Jeliot 3

5.1. *Internal Extensibility*

The development of Jeliot 3 is subject to several design principles. One of the original principles of Jeliot is to strive for extensibility, both internally and externally (Myller, 2004). Jeliot 3 features M-Code (Moreno, 2005), a linear transcription that represents the execution of an object oriented program, defining the program trace. An M-Code transcription is obtained by interpreting the program source code. The current version of Jeliot makes use of DynamicJava, a BSD licensed Java interpreter (see `http://koala.ilog.fr/djava/`) that has been used also in other projects (Allen *et al.*, 2002). The possibility to have an access to the source code of the interpreter was of great value to the development, as we were able to modify and build upon it. The modifications carried out in the Java interpreter were mainly to connect it with Jeliot 3, and to produce a textual representation of program trace, e.g., performing an assignment, an evaluation of a condition and similar.

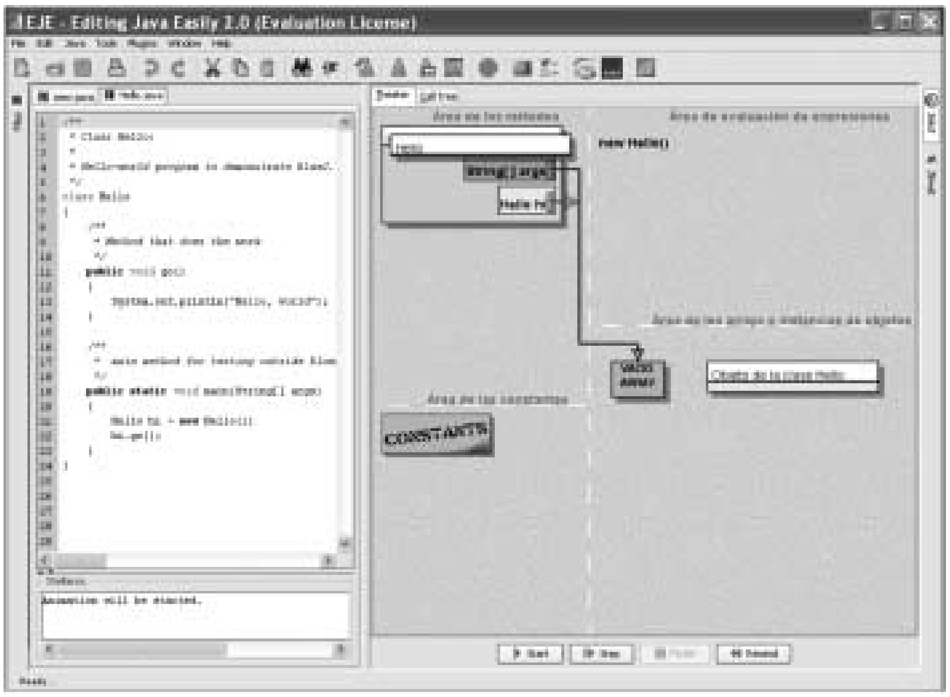
M-Code instructions are composed of an identification number to be referred later, an instruction code, indicating what kind of operation is performed, and one or more operands. It also carries information about the location in the corresponding source code. On the one hand, this open configuration allows for producing an animation of a program written in any programming language. To do so, the only requirement is to have an interpreter of such a language (e.g., C++) to produce the M-Code correctly and thus create an animation in Jeliot 3. Currently, there are several free interpreters for a wide range of programming languages available. At the moment, together with our collaborators we are working for versions to support C++, Pascal and Python.

On the other hand, the design of the M-Code allows the user-application to compose several different kinds of visualizations from a single M-Code stream. This means that one program execution can be used to generate several visualizations that illustrate different aspects of the program execution. It also makes it possible to distribute the visualization of the program execution to several users. As an example being currently implemented, besides the original view of the program execution based on the M-Code generated from the source code, Jeliot 3 offers also a dynamic call-tree view that is produced by using the same M-Code stream.

5.2. *External Plug-in Development*

Releasing Jeliot 3 as an open-source application has fostered many interesting collaborations leading to several innovations. One of those has been with the EJE team (`http://ilias.aifb.uni-karlsruhe.de/rku/`) at the University of Karlsruhe, Germany. This cooperation has been two-folded. On one side, the EJE team integrated Jeliot 3 into ILIAS (`http://www.ilias.de/ios/index.html`), an open source Learning Management System used at their institution. This project resulted in the ability of Jeliot 3 to be started from any lecture webpage, (Küstermann, 2005). On the application side, the EJE team develops a Java editor, EJE (Editing Java Easily). Jeliot 3 has been completely integrated as a visualization plug-in to EJE, see Fig. 2. The international cooperation between EJE and Jeliot 3 teams clearly benefited both tools. Jeliot 3 can make use of an improved environment, with an editor and other features, while EJE can visualize the code the user is creating.

The communication between EJE and Jeliot teams was mostly carried out by email. It included bug reports, comments, code modifications and code extensions to the Jeliot


EJE - Editing Java Easily 1.0 (Evaluation License)
Área de los métodos
Área de evaluación de expresiones
new Hello()
Área de los arrays e instancias de objetos
Área de las constantes
CONSTANTS
Objeto de la clase Hello
Start
Step
Pause
Rewind
Ready

```
/**
 * Class Hello:
 *
 * Hello-world program to demonstrate BlueJ.
 */
class Hello
{
    /**
     * Method that does the work
     */
    public void go()
    {
        System.out.println("Hello, World");
    }

    /**
     * main method for testing outside BlueJ
     */
    public static void main(String[] args)
    {
        Hello hi = new Hello();
        hi.go();
    }
}
```

```
Animation will be started.
```

Fig. 2. Screenshot of EJE running Jeliot 3 (on the right side).

main branch. To sum it up, due to the open source code and the extensible architecture, the cooperation between both parts has been made possible and yielded important innovations in e-learning tools.

Other plug-ins have been developed out of the main branch of Jeliot, namely the BlueJ plug-in and the JeCo tool. BlueJ is a widely-used integrated development environment that is oriented to Java students. It focuses on the graphical creation of classes by means of an UML notation and direct interaction with classes by creating objects and manipulating them in the object bench (Kölling, 2003). The plug-in architecture enables an interaction between BlueJ and Jeliot 3. Thus, it combines the standard UML notation and interaction with objects in the object bench used in BlueJ, like object creation and method calling, with the animation created by Jeliot visualization engine to represent such concepts.

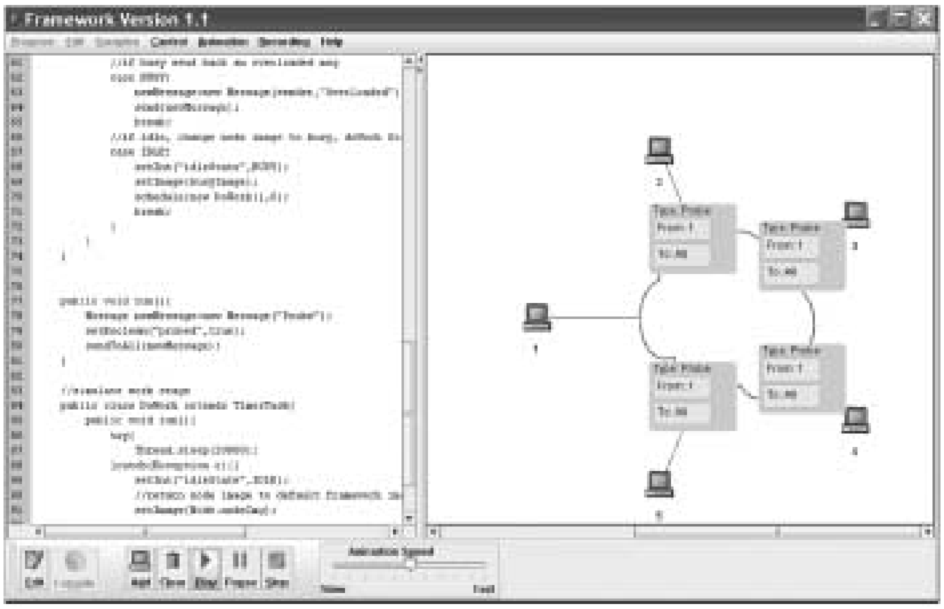
JeCo (Moreno *et al.*, 2004b) is a fully functional prototype that extends the concept of Woven Stories (Gerdt *et al.*, 2001) into a co-operative programming environment. Woven Stories is a co-authoring tool, where users can concurrently create text documents by adding nodes (sections) to a graph (document). The resulting graph then describes the document and its formation. JeCo makes use of the idea and features the novel concept of collaborative program visualization. It allows the user to visualize programs, send these visualizations to other users of JeCo, comment other users' programs and visualizations and chat with other users.

All these activities attribute to enriching the user learning and experience, letting users to approach their learning through the combination of integrated tools. However, the most important aspect is that the principle of openness lets researchers cooperate and create a community around the tools, connecting researchers, developers, teachers and students.

5.3. *New Tool Development*

Jeliot 3 releases have always been accompanied with the source code that built the releases. Thus, researchers, developers, and users can easily check the latest improvements done to the tool and they can apply them to their derived works.

As an illustration of the approach, Fionnuala O'Donnell, from the Trinity College at Dublin, Ireland, developed a simulation framework for teaching distributed systems concepts (O'Donnell, 2004), called FADA (Framework Animations of Distributed Algorithms, see Fig. 3 for a screenshot of FADA). It makes use of Jeliot 3 source code to implement the GUI and other parts of the tool. Thus, the authors of FADA could focus on the visualization side of distributed systems, and make use of a novice oriented GUI of Jeliot 3. An important advantage of having different visualization tools sharing the same GUI is that when users encounter the same GUI again they can expect the same functionality as with the previous tool. This is an important fact, as it has been often claimed that novice users have to be specifically taught to use their new environments (e.g., Ben-Ari, 2001). Therefore, this approach can reduce the initial problems the novice students have to encounter. On the example FADA, we can observe that the reuse of the open-source environment resulted in a new tool with different pedagogical purposes.



```
[illegible]
```

Fig. 3. FADA screenshot.

6. Community around Jeliot

One of the main goals of the on-going development of Jeliot is to create a community of users and developers around the tool. Currently, most of the users are Finnish and Israeli students and teachers. Some of the user groups, such as young students and high school teachers in Israel and senior students in Finland, feel Jeliot as an integral part of their learning; they have developed a sense of ownership of the tool (Squires, 1999). Such feeling fosters learning through the tool, however, it has also been reported to produce dependency, and users can feel confused when the tool is not present.

Mailing lists and forums are set up for building and connecting different institutions using Jeliot 3 and we have received valuable feedback, ideas and propositions to the further development. However, we have seen that there are improvements needed to get students and teachers more involved and motivated. Different reasons for this behavior can be considered. For example, as most of the student-users are Finnish and Israeli who may not be fluent in English, they may feel insecure about expressing their thoughts in a foreign language.

7. Conclusion

Jeliot 3 is an open and modular program visualization tool targeted especially to novice programmers and students. We have shown how the open-source philosophy and design decisions yielded into extensibility in different contexts resulting into new tools, and how these extensions have benefited the development of the tool itself. Jeliot 3 itself is put together from open source software, too, demonstrating the power of open source development for real educational learning tools. Comparing with other available program visualization tools, we consider our approach as rather innovative.

We presented three platforms that use Jeliot 3 as a plug-in to deliver program visualizations to the users of the systems. Two of these systems are in wide use and thus bring Jeliot 3 available to large audiences. Moreover, there is a tool derived from Jeliot 3 showing how open source licensing can support efforts of other developers.

The development of the Jeliot family has been driven by academic research that has guided and given new directions for the further development. We have and will be carrying out several experiments with Jeliot 3 to validate its usability and efficiency in the context of introductory programming both in face-to-face as in on-line courses. So far the results have been positive and we expect to extend the use of Jeliot into collaborative learning where group processes should be supported as well as into distance learning where adaptivity to the users' needs is crucial. Our work is supported by the community of users and researchers formed around Jeliot and the feedback and suggestions have helped us to enhance Jeliot 3 to better suit all the different needs of our users. We hope to see the community evolving into the direction of a true learning and collaborative community where the roles of people are changing, and where a developer becomes a learner and learner a developer.

References

Allen, E., R. Cartwright and B. Stoler (2002). DrJava: a lightweight pedagogic environment for Java. In *Proceedings of the 33rd SIGCSE technical symposium on Computer Science Education*. ACM Press, pp. 137–141.

Bednarik, R., A. Moreno, N. Myller and E. Sutinen (2005). Smart program visualization technologies: planning a next step. In *Proceedings of the 5th IEEE International Conference on Advanced Learning Technologies* (*ICALT 2005*), Kaohsiung, Taiwan. IEEE Computer Society, pp. 717–721.

Ben-Ari, M. (2001). Program visualization in theory and practice. *Informatik. Informatique*, **2**, 8–11.

Ben-Ari, M., N. Myller, E. Sutinen and J. Tarhio (2002). Perspectives on Program Animation with Jeliot. In S. Diehl (Ed.), *Software Visualization*, *Lecture Notes in Computer Science*, vol. 2269. Springer-Verlag, Dagstuhl, pp. 31–45.

Ben-Bassat Levy, R., M. Ben-Ari and P.A. Uronen (2003). The Jeliot 2000 program animation system. *Computers and Education*, **40**(1), 15–21.

Boroni, C.M., T.J. Eneboe, F.W. Goosey, J.A. Ross and R.J. Ross (1996). Dancing with DynaLab: endearing the science of computing to students. *SIGCSE Bulletin*, **28**(1), 135–139.

Detienne, F. (2002). *Software Design – Cognitive Aspects*. Springer-Verlag, London.

Ericsson, K.A. (2003). The acquisition of expert performance as problem solving: Construction and modification of mediating mechanisms through deliberate practice. In J.E. Davidson and R.J. Sternberg (Eds.), *Problem Solving*. New York, Cambridge University Press, pp. 31–83.

Gerdt, P., P. Kommers, C. Looi and E. Sutinen (2001). Woven stories as a cognitive tool. In *Cognitive Technology*, *Lecture Notes in Artificial Intelligence*, Vol. 2117. Springer-Verlag, pp. 233–247.

Gestwicki, P., and B. Jayaraman (2005). Methodology and architecture of JIVE. In *Proceedings of the 2005 ACM Symposium on Software Visualization* (*SoftVis '05*). ACM Press, New York, NY, pp. 95–104.

Hoc, J.-M., T.R.G. Green, R. Samurcay and D.J. Gilmore (Eds.) (1990). *Psychology of Programming*. Academic Press.

Hundhausen, C.D., S.A. Douglas and J.T. Stasko (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, **13**(3), 259–290.

Korhonen, A., L. Malmi, P. Silvasti, V. Karavirta, J. Lönnberg, J. Nikander, K. Stålnacke and P. Ihantola (2004). *Matrix – A Framework for Interactive Software Visualization*. Laboratory of Information Processing Science, Department of Computer Science and Engineering, Helsinki University of Technology. TKO-B 154/04, Research Report.

Küstermann, R., D. Ratz and D. Seese (2005). Effektive Java-Grundausbildung unter Einsatz eines Learning Management Systems und spezieller Werkzeuge. In *Proceedings of the INFOS' 05*. pp. 10.

Kölling, M., B. Quig, A. Patterson and J. Rosenberg (2003). The BlueJ system and its pedagogy. *Journal of Computer Science Education*, **13**(4).

Lahtinen, S.-P., E. Sutinen and J. Tarhio (1998). Automated animation of algorithms with Eliot. *Journal of Visual Languages and Computing*, **9**(3), 337–349.

Moreno, A. (2005). *The Design and Implementation of Intermediate Codes for Software Visualization*. Department of Computer Science, University of Joensuu, Joensuu, Finland.
http://cs.joensuu.fi/jeliot/files/Andres_thesis.pdf

Moreno, A., N. Myller, E. Sutinen and M. Ben-Ari (2004a). Visualizing Programs with Jeliot 3. In *Proceedings of the International Working Conference on Advanced Visual Interfaces* (*AVI 2004*). Gallipoli, ACM Press, pp. 373–376.

Moreno, A., N. Myller and E. Sutinen (2004b). JeCo, a collaborative learning tool for programming. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing* (*VL/HCC'04*). Rome, IEEE Press, pp. 261–263.

Myller, N. (2004). *The Fundamental Design Issues of Jeliot 3*. Master's thesis. Department of Computer Science, University of Joensuu, Joensuu, Finland.
ftp://cs.joensuu.fi/pub/Theses/2004_MSc_Myller_Niko.pdf

O'Donnell, F. (2004). *A Simulation Framework for the Teaching of Distributed Systems Concept*.
https://www.cs.tcd.ie/Fionnuala.ODonnell/Framework/index.htm

Rössling, G., and B. Freisleben (2002). ANIMAL: A system for supporting multiple roles in algorithm animation. *Journal of Visual Languages and Computing*, **13**(3), 341–354.

Sajaniemi, J., and M. Kuittinen (2003). Program animation based on the roles of variables. In *Proceedings of the 2003 ACM Symposium on Software Visualization*. ACM Press, pp. 7–16.

Squires, D., and J. Preece (1999). Predicting quality in educational software: evaluating for learning, usability and the synergy between them. *Interacting with Computers*, **11**(5), 467–483.

Stasko, J.T. (1990). Tango: A framework and system for algorithm animation. *IEEE Computer*, **23**(9), 27–38.

Sutinen, E., J. Tarhio and T. Teräsvirta (2003). Easy algorithm animation on the Web. *Multimedia Tools and Applications*, **19**(2), 179–184.

R. Bednarik has received MSc (2002) and PhLic (2006) degrees from the Department of Computer Science, University of Joensuu, Finland. At the moment he is completing a PhD project in the field of eye-movement tracking methodologies. In particular, his research interests include HCI, psychology of programming, usability, program visualization and computer science education. He has published several papers in reviewed journals and international conferences.

A. Moreno is a PhD student at the University of Joensuu, Finland, since May 2005. He received his master's degree from the Polytechnic University of Madrid, Spain. His master's thesis developed an intermediate code for program animation, and was jointly supervised by professors Erkki Sutinen and Mordechai Ben-Ari. He is currently researching on program visualization and animation for novices, focusing on how to make current visualization tools aware of the personal differences of the users. Having published in conferences such as ACM ITiCSE and IEEE ICALT, he is an active member of the Algorithm Animation community. He has also taken part in several working groups at ITiCSE.

N. Myller received his BSc in 2003 and MSc in 2004 both from the Department of Computer Science at University of Joensuu in a record time of 2.5 years from starting. Currently, he is studying for his PhD under supervision of prof. Erkki Sutinen (University of Joensuu) and prof. Mordechai Ben-Ari (Weizmann Institute, Israel) and the expected graduation is in 2007. His research interests lie in the fields of visualization and concretization technologies, CSCL, information retrieval, computer ethics and adaptive systems. He has published more than 30 papers in international journals and conferences.

Skirtingi atvirosios programos, vizualizavimo priemonės „Jeliot 3“, panaudojimo būdai

Roman BEDNARIK, Andrés MORENO, Niko MYLLER

Straipsnyje supažindinama su atvirąja vizualizavimo priemone „Jeliot 3“. Aptariami dizaino principai ir ideologija, leidusi populiarėti šiai puikiai elektroninio mokymo priemonei bei dar keletui kitų su ja susijusių aplinkų. Būtent, greta „Jeliot 3“ straipsnyje aptariamos ir trys kitos aplinkos: „BlueJ“, „EJE“ ir „JeCO“, naudojančios „Jeliot 3“ kaip papildinį, skirtą programos tekstui vizualizuoti. Taip pat aptariama ir FADA aplinka, kuri sukurta remiantis „Jeliot 3“, tačiau naudojama skirtingiems pedagoginiams tikslams. Pastebėtina, jog jau yra susibūrusi šių projektų naudotojų ir tobulintojų bendruomenė, tad „Jeliot 3“ priemonė nuolat tobulinama remiantis tiek akademiniais tyrimais, tiek bendruomenės pateikiamomis pastabomis ir pastebėjimais. Straipsnyje aprašoma programa gretinama su kai kuriomis kitomis šiuolaikinėmis priemonėmis, supažindinama su kai kuriais itin sėkmingais šios programos panaudojimo atvejais.