Proceedings of the Sixth Program Visualization Workshop

Guido Rößling (Ed.)
TU Darmstadt, Germany—June 30, 2011
Technical Report Number: TUD-CS-2011-0153

TECHNISCHE UNIVERSITÄT DARMSTADT

TECHNISCHE UNIVERSITÄT DARMSTADT

Contents

1 Visualization and Interaction with Multiple Levels of Detail in Greedy Algorithms

J. Ángel Velázquez-Iturbide
Dpto. Lenguajes y Sistemas Informáticos I, Escuela Técnica Superior de Ingeniería Informática
Universidad Rey Juan Carlos
Móstoles, Madrid, 28933

angel.velazquez@urjc.es

1.1 Introduction

Instructors and researchers have been interested in algorithm visualization (AV) for the last three decades. As a consequence, a large number of AV systems (and built-in animations) are currently available for educational use (Shaffer et al., 2010).

If AV systems had been designed according to given educational goals, it would be easy to state the kind of tasks they can support. However, hardly any AV system has been designed this way. Alternatively, we may deduce their implicit goals by examining the literature (Hundhausen et al., 2002) (Urquiza and Velázquez, 2009). An analysis reveals that most AV systems are designed to aid students in understanding algorithm behavior, with some systems also supporting the gathering of performance measures or the comparison of alternative algorithms.

Our hypothesis is that the use of explicit, clear pedagogical goals increases the quality and effectiveness of an AV system. Firstly, its visualization and interaction requirements can be more easily identified. Secondly, instructional activities can be checked for alignment with the educational goals of our system. Finally, broadening the range of intended educational goals may result in broadening the range of features supported by the AV system.

In this paper we exemplify this approach for the active learning of greedy algorithms. The statement of educational goals for greedy algorithms allowed us to design an experimental method; furthermore, we developed an interactive assistant, called GreedEx, which supports the method for several problems. In this paper we describe the data necessary to reason about optimality in greedy algorithms, as well as visualization and interaction facilities designed for this task. Further information can be found elsewhere (Velázquez and Pérez, 2010).

The structure of the paper follows. The second section sets the background of this work, briefly describing our experimental method and the interactive assistant GreedEx. Section 3 describes the visualization of greedy algorithms at three levels of detail, as well as interaction techniques aimed at making them coherent and flexible. In the fourth section, we describe our experience and the results of several usability evaluations. Finally, we discuss related work, state our conclusions and outline current and future work.

1.2 Background

We have developed the GreedEx interactive assistant (Velázquez and Pérez, 2009) to support an active learning method. It assists in several problems that can be solved with greedy algorithms, including the activity selection problem and the knapsack problem. The problems supported share functionality and user interface except in problem-specific details, such as data visualization. To be specific, we illustrate this paper with the activity selection problem.

Figure 1.1 shows the user interface of GreedEx. It shows an intermediate state of execution for the selection of activities problem in increasing order of index. Three areas can be clearly distinguished. The top area is the visualization panel, which graphically displays data. The lower left area is the theory panel, which consists of two tabs: the problem tab contains the problem statement, and the algorithm tab, its code in Java pseudocode (visible in the figure). Finally, the lower right area is the table panel, with four tabs containing the input data table, the results table, the summary table, and the abridge table (see the figure).

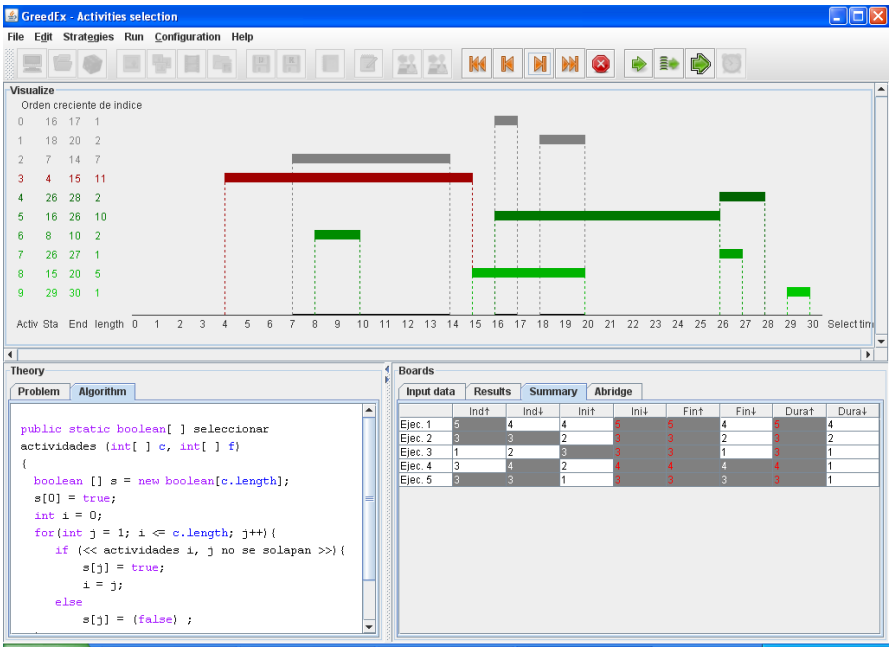


Figure 1.1: User interface of the interactive assistant GreedEx

When the user launches the application, only the theory panel hosts visible contents. Its two tabs can be read to understand the problem. Afterwards, the user may generate input data from three sources: interactively, randomly or from a file. Some limits are imposed over data size to keep data manageable for learning and for visualization.

Input data are displayed in the visualization panel. The user may then choose any selection function from those available at the application. Every time the user selects a selection function, she may execute it flexibly with four execution/animation controls: step forward, full forward, step backward, and rewind. As animation controls are clicked, the visualization is updated accordingly. After a selection function has been completely executed, the results of the algorithm are stored in the results table. The results are also stored in the summary table. The abridge table shows the optimality percentage of the results obtained by each selection function.

Additional facilities are provided for faster experimentation: executing all the selection functions, executing a subset of the selection functions, and generating and executing all the selection functions over a very high number of input data.

1.3 Multiple levels of detail in visualizations

The experimental method described in the previous section generates a large amount of data. In this section, we describe the structuring and visualization of data in three levels of detail, as well as connection among those levels.

1.3.1 Single set of input data, single selection function

The smallest piece of experiment that can be carried out consists in executing the greedy algorithm for particular input data and a particular selection function. The learning goals of this activity are:

- Comprehension level: Understanding the algorithm behavior with the given input data and selection function.
- Analysis level: Analyzing the effect of the selection function.

The assistant gathers and displays data to support these learning goals. The visualization panel is the main means to support comprehension, while the results table is the main means to support analysis.

The user may control the pace of execution with the animation controls (both forwards and backwards) and may observe the execution state in the visualization panel. At each step, the following information must be displayed:

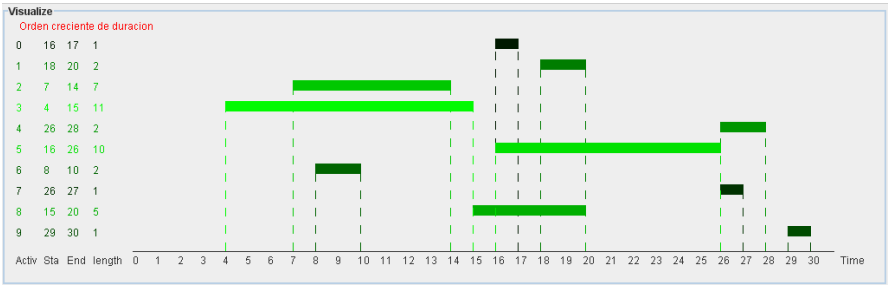


Figure 1.2: Visualization of activities in increasing order of duration

- Input and output data.
- Status of candidates: active, discarded, selected.
- Relative order of active candidates.

This information is shown in the visualization panel as an algorithm animation in a problem-specific format. Upon termination of the algorithm execution, its outcome is updated in the results and summary tables.

Figure 1.1 illustrates a visualization for the activity selection problem. The activities are displayed in a bidimensional matrix with a temporal axis. Each activity is displayed as an horizontal bar, ranging between its start time and its end time.

Colors are used to make clear the status of each activity during the execution of the greedy algorithm. Activities are initially displayed in green color. Every time an activity is selected, it is re-colored either in gray (if it is inserted into the solution set) or in red (if it overlaps with an activity previously selected).

Color tones allow sharing the visualizations for all the selection functions, while reporting the relative order of the candidates for every selection function. The criterion we have adopted is that the darkest tones are assigned to the candidates that are selected the first according to the active function selection.

In Figure 1.1, the activities are selected according to increasing order of index. Notice that the first four activities were considered (at the top of the display) and there are six pending activities. The tones of these activities become lighter as we descend in the visualization, as corresponds to this selection function.

Figure 1.2 contains an alternative assignment of tones for increasing order of duration as the selection function. The shortest activities are colored in dark, activity 0 being the first one. The longest activities are colored in light, activity 3 being the longest.

1.3.2 Single set of input data, multiple selection function

A higher level in the experiment consists in comparing the results of the different selection functions over the same input data. The learning goals of this level are:

- Analysis level: Analyzing the effect of the selection functions over the same input data.
- Evaluation level: Evaluation of the outcomes of the different selection functions and decision on which are optimal for this execution.

The results of the different selection functions must be shown simultaneously in order to compare them. In this case, a table is an adequate, general means (see Figure 1.3) to display information relevant for analysis and evaluation. Its format is similar for all the interactive assistants. The table shows rows corresponding to the selection functions actually executed for the current data set.

Some columns also are common for any problem:

- Selection function (called strategy).
- Candidates selected under such a function.
- Outcome produced by these candidates.

Figure 1.3 shows that these columns are sufficient for the activity selection problem. In other problems, additional columns may be useful to better understand the selection of candidates and their effect.

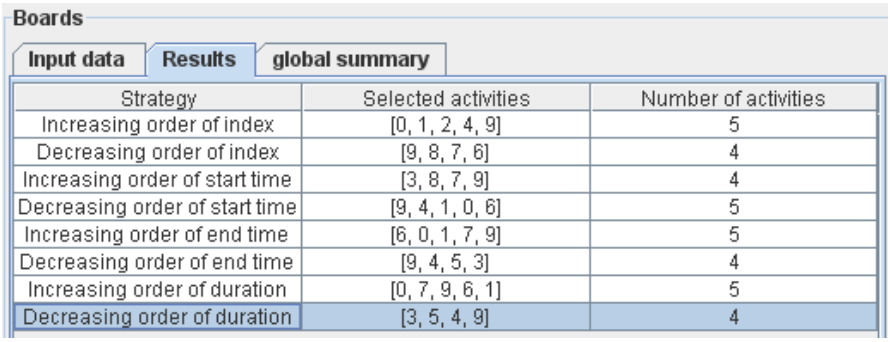


Figure 1.3: Table of results for a single data set for the activity selection problem

1.3.3 Multiple set of input data, multiple selection function

At the highest level of the experiment, the selection functions must be compared for optimality. The learning goals of this level are:

- Evaluation level: Evaluate the outcomes of selection functions and decide which are optimal.

The user should be able to visually recognize the selection functions that always yield optimal results. These data are displayed in the summary table, which has a very simple structure. Each row stores the results obtained for one data set. Each column stores the results obtained for a selection function. Each cell may contain either a dash or a numeric value, depending on whether its corresponding selection function was or was not applied to its corresponding input data.

In order to make easier to the user to identify optimal solutions, cells that contain optimal outcomes for a data set are highlighted in gray. In addition, the columns that contain optimal values in all its rows highlight the font color of values in red.

Figure 1.1 shows this table in its lower right panel. Notice that there are three selection functions that have always produced optimal results in this five-iteration experimentation.

1.3.4 Connection between levels

The three levels of detail supported are aimed at different learning goals. However, generation and inspection of the data gathered are not always done in sequence. In general, the user examines data in forward and backward directions. We briefly mentioned this fact for our first level of detail: the user will typically need to backtrack at some step of the animation in order to understand a previous step. This capability is provided by two backward animation controls.

This need to watch past results also arises at higher levels of detail. For instance, the user may want to watch past behavior of a selection function over the current data set. Furthermore, she may want to watch that behavior over a past data set.

In order to support such exploratory navigation, the user must be able to backtrack using adequate affordances. Fortunately, a very intuitive affordance for these connections consists in simply clicking on a row or on a cell. Thus, if the user selects a row in the results table, the visualization panel is updated with the final state resulting of applying the corresponding selection function. The user may simply watch in the visualization panel the effect of applying this selection function, or she may freely navigate in the animation.

Alternatively, the user may select a cell in the summary table. The visualization panel is then updated with the result of applying the corresponding selection function to the corresponding input data. Again, the user may freely animate the visualization.

Figure 1.1 illustrates this connection between levels. Although five input data have been generated and executed, the display in the visualization panel corresponds to the first input data under selection in increasing order of index. The user clicked the corresponding cell in the summary table, then rewound the visualization and advanced four steps forward.

Table 1.1: General usability results

Question	1^{st} sess.	2^{nd} sess.	3^{rd} sess.	4^{th} sess.
Easy to use	4.54	4.64	4.42	4.38
It has aided me in analyzing the effect of each selection function	4.20	4.50	4.36	4.19
It has aided me in identifying optimal selection functions	4.55	3.60	4.04	4.04
Overall, I liked it	4.25	4.27	4.32	4.19

Table 1.2: Usability results for visualization elements

Element	1^{st} sess.	2^{nd} sess.	3^{rd} sess.	4^{th} sess.
Visualization panel	4.28/4.03	4.18	4.46	4.00
Algorithm animation	-	-	4.33	4.19
Results table	4.18	4.18	4.36	4.31
Summary table	-	-	4.39	3.88

1.4 Experience and evaluation

The GreedEx assistant is the result of several years of development and enhancement. We started from an initial specification based on pedagogical goals. We also reviewed illustrations in textbooks in order to adopt the best graphical representations. Thus, our visualizations are variants of figures in Sedgewick's book for the knapsack problem (Sedgewick, 2002) and in Cormen et al.'s for the activity selection problem (Cormen et al., 2001). In the latter, we were able to simplify the figure layout thanks to the time dimension provided by the animation.

Several assistants, with similar features, were developed (Velázquez and Pérez, 2009). They were also evaluated in two ways:

- Expert evaluations. They were repeatedly performed by the instructor to guarantee conformance to the specifications as well as usability and adequacy to actual use in algorithms courses.

- Evaluations with students in lab sessions. They gave us feedback about user acceptance, evaluation of the main features and elements, and open suggestions. We also graded students' findings with the assistants.

Finally, the assistants were integrated into a single system, namely GreedEx. In the following subsections, we summarize our experience of use as well as the characteristics and results of students' usability evaluations.

1.4.1 Experience

The interactive assistants were used during the last three academic years in the compulsory course "Design and Analysis of Algorithms" and in the optional course "Advanced Data Structures and Algorithms", both placed at the third year of the Degree of Computer Science at our university. They were used as follows:

- Academic year 2007-2008, first term. The assistant for the knapsack problem was used in an assignment on greedy algorithms.

- Academic year 2007-2008, second term. The assistant for the activity selection was used in an assignment on greedy algorithms.

- Academic years 2008-2009 and 2009-2010, first term. The teacher used in the classroom the assistants for the knapsack problem and for the minimum cost spanning tree, and he used the assistant for the activity selection problem in the assignment on greedy algorithms.

1.4.2 Usability evaluations

We held four usability evaluation sessions, one per assignment session. Each session was two hours long. Students were given the assignment statement, the assistant, a report template and an opinion questionnaire.

Students had to perform three tasks:

1. Use the assistant to determine which of the selection functions offered are optimal.
2. Fill and deliver a short report on their findings, using a given template.
3. Fill and deliver an opinion questionnaire about the assistant.

Some additional features of the sessions follow:

1. January 2008. Knapsack problem. 46 participants. The assignment and the report were done in pairs, but the questionnaire was answered individually.
2. May 2008. Activity selection problem. 13 participants. All student work was done individually.
3. January 2009. Activity selection problem. 31 participants. Similar to the first session.
4. November 2009. Activity selection problem. 27 participants. Similar to the first session.

The opinion questionnaires used in these sessions consisted of open questions and quizzes, with values in a Linkert scale ranging between 1 (very bad) to 5 (very good).

The results of the four sessions were very positive. Table 1.1 contains the mean values obtained for general usability issues. Notice the very positive marks obtained in all the evaluations. We also call the attention to the fact that the assistants were a bit more complex in each evaluation with respect to the previous one.

We also asked students their opinion about individual elements of the assistants. Table 1.2 contains the results for those elements involving visualizations. Notice that some elements were not graded in some evaluations. This was due to the fact that they were not included in the questionnaire or that the version used for the session did not support such a feature yet. The two values given to the visualization panel in the first session are due to the fact that the assistant for the knapsack problem had two subpanels (one for input data and other for output data).

The evolution of the values obtained is not always smooth. The first and second evaluations yielded similar values, even though the assistant used in the second session incorporated enhancements, based on the results of the first session. The third session included further improvements, which resulted in better scores. However, the fourth session delivered poorer results, even though the same version of the assistant was used. This was mainly due to some problems we experienced with the lab installation.

1.5 Related work

The work described in this paper is related to several issues that we address in this section.

1.5.1 Properties of algorithms, learning goals and visualization

As we cited in the introduction, hardly any AV system is designed upon explicit educational goals. However, an analysis of the literature (Hundhausen et al., 2002) (Urquiza and Velázquez, 2009) shows that most algorithm animations are implicitly oriented to algorithm comprehension. Some interaction facilities are common in animations aimed at comprehension (Naps et al., 2003): pop-up questions, backwards animation, entry of input data, etc. Correctness is the main property supported, efficiency is sometimes considered and, as far as we know, optimality is never explicitly addressed.

Deeper learning can be achieved by aiming at higher learning goals (Bloom et al., 1956), e.g. analysis or evaluation. Although some of the techniques listed above are useful for effective analysis, more flexible interaction is required in general. A prototypical example is the ZStep 95 system, with its facilities designed to assure that the user "experiences immediacy" (Ungar et al., 1997). Bloom's level of evaluation has often been addressed to compare the efficiency of alternative algorithms for a given problem, e.g. sorting algorithms (Brown and Sedgewick, 1983).

1.5.2 Visualizations in algorithm animation

Our visualizations have similarities with other works. We have cited in the previous paragraph the "multiple algorithms" technique. The comparison among several algorithms is typically left to the visual perception of the user, complemented with brute performance data (Brown and Sedgewick, 1983). The different levels of detail in our system resemble the "multiple views" technique. We emphasize that our levels of detail are clearly characterized.

A few systems have structured their visualizations at different levels of abstraction. In the HalVis system (Hansen et al., 2002), every algorithm supported is animated at three levels of abstraction: an everyday metaphor, one related to code, and one over large input data. The "Algorithms in Action" system (Stern et al., 1999) supports up to three levels of pseudocode, each one showing a coherent visualization. The guidelines used in this work for abstraction separation are looser. Finally, an "instructor's guide" has been proposed with advice on how to structure animations (Velázquez et al., 2008). It also considers abstraction but, once the animation has been generated, it is not allowed to change the level of abstraction.

1.5.3 Interaction and information visualization

We may characterize our proposal by referring to an external, comprehensive framework. We adopt a taxonomy of interaction techniques in information visualization (Yi et al., 2007) that we found useful in the past to characterize the SRec animation system (Velázquez and Pérez, 2010).

Our assistant GreedEx includes abstract/elaborate (i.e. change of detail) and exploration (i.e. animation) facilities. However, their most remarkable interaction categories, for the purpose they were designed, are:

- "Rearrangement" category. Each interactive assistant provides a "general" greedy algorithm and a "general" visualization. The effect of each selection function is shown with color tones, which implicitly sort the candidates.
- "Connect" category. The user often needs switching among the three levels for inspection. By clicking on the tables cells, she brings its data to the visualization panel.

1.6 Conclusions and future work

We have shown a design of visualization and interaction based on learning goals for a class of algorithms, namely greedy algorithms. There are four issues that, as far as we know, make this work unique. Firstly, explicit learning goals were stated before developing the system: comprehension, analysis, and evaluation. Secondly, the visualizations addressed algorithmic optimality. Thirdly, the design of the visualization was structured in three levels of detail, and complemented with certain interaction techniques (mainly, rearrangement and connection). Although this design was implemented for particular algorithms, it is of general applicability to any greedy algorithm. Finally, the assistants have been evaluated for usability (with positive results) and regularly enhanced for several years.

Comprehensive information about the GreedEx system can be found at the following URL: http://www.lite.etsii.urjc.es/greedex/

We are currently working in two directions. Firstly, we are studying who to extend our experimental method to support collaborative learning. Secondly, we are studying how to widen the scope of GreedEx to fully support a class of combinatorial problems, instead of a fixed collection of problems.

Acknowledgments. This project is supported by project TIN2008-04103/TSI of the Spanish Ministry of Science and Innovation.

Bibliography

B. Bloom, E. Furst, W. Hill, and D. R. Krathwohl. *Taxonomy of Educational Objectives: Handbook I, The Cognitive Domain*. Addison-Wesley, 1956.

M. H. Brown and R. Sedgewick. Techniques for algorithm animation. *IEEE Software*, 2(1):28–39, 1983.

T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 2001.

S. Hansen, D. Schrimpsher, and N. H. Narayanan. Designing educationally effective algorithm animations. *Journal of Visual Languages and Computing*, 13:291–317, 2002.

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

T. Naps, G. Roessling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, and J.Á. Velázquez. Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin*, 35(2):131–152, 2003.

R. Sedgewick. *Algorithms in Java*. Addison-Wesley, 2002.

C. A. Shaffer, M. L. Cooper, A. J. D. Alon, M. Akbar, M. Stewart, S. Ponce, and S. H. Edwards. Algorithm visualization: The state of the field. *ACM Trans. Computing Education*, 10(3), 2010.

L. Stern, H. Sondergaard, and L. Naish. A strategy for managing content complexity in algorithm animation. In *Proceedings of 4th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 1999)*, pages 127–130, 1999.

D. Ungar, H. Lieberman, and C. Fry. Debugging and the experience of inmediacy. *Communications of the ACM*, 40(4):38–43, 1997.

J. Urquiza and J.Á. Velázquez. A survey of successful evaluations of program visualization and algorithm animation systems. *ACM Trans. Computing Education*, 9(2), 2009.

J. Á. Velázquez and A. Pérez. Active learning of greedy algorithms by means of interactive experimentation. In *Proceedings of 14th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2009)*, pages 119–223. ACM Press, New York, 2009.

J. Á. Velázquez and A. Pérez. Infovis interaction techniques in animation of recursive programs. *Algorithms*, 3:76–91, 2010.

J. Á. Velázquez, D. Redondo, C. Pareja, and J. Urquiza. An instructorŠs guide to design web-based algorithm animations. *Lecture Notes in Computer Science*, 4823/2008:440–451, 2008.

J. S. Yi, Y. A. Kang, and J. T. Staskoand J. A. Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):224–231, 2007.

2 Visualization of the syntax tree in compilers construction courses. Educational evaluations

Francisco J. Almeida-Martínez, Jaime Urquiza-Fuentes, Antonio Pérez-Carrasco
Departamento de Lenguajes y Sistemas Informáticos I. Escuela Técnica Superior de Ingeniería Informática. Universidad Rey Juan Carlos. Madrid

{francisco.almeida,jaime.urquiza,antonio.perez.carrasco}@urjc.es

2.1 Introduction

The syntax directed translation (SDT) is one of the most complex aspects of subjects as *Language Processors* or *Compilers*. The correct application of the SDT requires the appropriate comprehension of the previous stages, in particular, in the syntax stage. In this stage, it is extremely important the syntax tree, which is the base for next stages.

Numerous works show that the use of visualizations help to improve the learning process (Hundhausen et al., 2002). In this context, the visualization of the SDT can help its comprehension. As the SDT uses the syntax tree structure, resulting from the syntax analysis, it is necessary to create visualizations of the syntax tree and make sure that they can help to learn. As it is described in the section 2.2, there exists a large number of tools that allow to visualize and animate some aspects of the compilation process. However, these tools have two limitations. On the one hand, they depend on an particular generation tool. On the other hand, the number of views given by these tools is limited. According to these characteristics, if it is necessary to visualize one aspect which the current tool does not display, it is necessary to change both the visualization and generation tools, which could prevent from using this kind of tools (Levy and Ben-Ari, 2007) and difficult the learning of students.

In this work we present VAST (VisuAlizer of the Syntax Tree) and different evaluations with educational aim. VAST allows to display the syntax tree and other structures of the syntax analysis process. VAST has been implemented following a generic approach, this means that it has not dependencies with the automatic generation tools. Besides, we have developed a design which allows to easily add new views of the compilation process. Apart from the evaluations presented here, VAST has been evaluated in Almeida-Martínez and Urquiza-Fuentes (2009).

The rest of the communication is structured as follow. In section 2 we describe the related works. In section 3 we show the main characteristics of VAST. In section 4, 5, 6 and 7 we detail the different educational evaluations performed. Finally, in section 8 we present the conclusions and future line of work.

2.2 Related work

From the introduction of the compilers construction in the Computer Science, it has been observed that students have important difficulties to understand the theory underlaying (Chanon, 1975). In these subjects it is necessary to have into account two important factors: the available time and the students' demotivation. According to the available time, it becomes a critical aspect because teachers have to adapt the materials to a small period of time (Griswold, 2002). The demotivation of students normally is caused because just a small number of them will develop compilers professionally (Demaille, 2005).

There exists two opposite methodological approaches to teach compilers. On the one hand, there exists a methodology which have a theoretical aim, without practical exercises. On the other hand, there exists another methodology with a complete practical aim. Given the characteristics of these materials, none of these approaches is appropriate to teach them (Aho, 2008). The traditional methodology consists in theoretical explanations with practical exercises, normally their main objective is to build a compiler for an specific language. In this methodology we can distinguish two approaches. On the one hand we have those approaches which require to implement the compiler manually, not using generation tools. On the other hand we have other approaches which allow to use generation tools, i.e COOL (Aiken, 1996).

In order to improve the students' motivation there are other approaches which adapt the material and the available time, i.e Ten MiniLanguages (Ledgard, 1971). These new approaches can be combined with visualization tools, which can have both theoretical and practice aim. JFLAP (Rodger et al., 2007) is the most representative tool with theoretical aim.

Those tool with practical aim can be classified in three groups. The tools in one group do not allow to generate parsers because they are developed to recognisee a specific language, i.e. ICOMP (Kristy et al., 1988). Other group is formed by those tools which can generate parsers but their visualizations are oriented to a profesional user, i.e VCOCO (Resler and Deaver, 1998). Finally, the rest of the tools generate more elaborated visualizations, i.e ANTRLWorks (Bovet and Parr, 2008). However, all of them depend on an own notation or a specific parser generator. This approach makes more difficult their educational use.

2.3 Syntax analysis visualization

In this section we present VAST (VisuAlizer of the Syntax Tree) (Almeida-Martínez et al., 2009). VAST allows to perform the visualization of the syntax tree with a generic approach, independently from the generation tool used. Firstly we describe the generation of visualizations with VAST and its user interface. Finally we describe the visualization of the syntax tree, the animation of its building process and the visualization of the syntax error recovery.

2.3.1 Generation of visualizations and the user interface

VAST is divided in two modules: VASTapi and VASTview. VASTapi is in charge of obtaining the necessary information during the analysis to build the visualizations. To this end, it offers an API to interpret the actions executed during the analysis. Besides it has to include the calls to the API in the parser specification (annotation). VASTview is in charge of performing all the visual representations of VAST, so it has to interpret the information created by VASTapi.

The user interface of VAST is divided in 6 different zones. In Fig. 2.1 it is shown an example of the general user interface. In the middle it is the syntax tree. In the top right corner it is shown the input stream used for the analysis. Finally, under the syntax tree, it is shown the global view, the parser's stack, the actions performed during the analysis and the grammar used.

2.3.2 Animating the parsing process and syntax error recovery

The animation of the analysis process allows to watch how the syntax tree is built while the input stream is processed. As VAST is compatible with both LL and LR parsers, the user can see how both kinds of trees are built. In order to animate the construction process, VASTview has two types of reproduction control: step by step and automatic.

VAST can display 3 different strategies of syntax error recovery (Almeida-Martńez et al., 2010): phrase level, error productions and panic mode.

The phrase level error recovery is based in insertions/deletions from the input stream. As the ANTLR generator works with this kind of error recovery, we have added this functionality to VAST. We have modify the annotation process of VASTapi.

The error recovery using error productions has been implemented for ANTLR and Cup tools. This strategy works executing a normal production as it were an error one, so when a production of this type is used, it should be reported as an error. Note that this strategy depends on the programmer decisions. In case of VAST, we have added a new method to VASTapi which allows to distinguish between normal productions and error ones.

Finally, the panic error recovery has been implemented just for Cup. This strategy consists in defining synchronization points. When an error occurs, the input stream is ignored until a synchronization point is found.

2.4 Evaluation I

The version of VAST used in this evaluations allowed to display the syntax tree and its construction process.

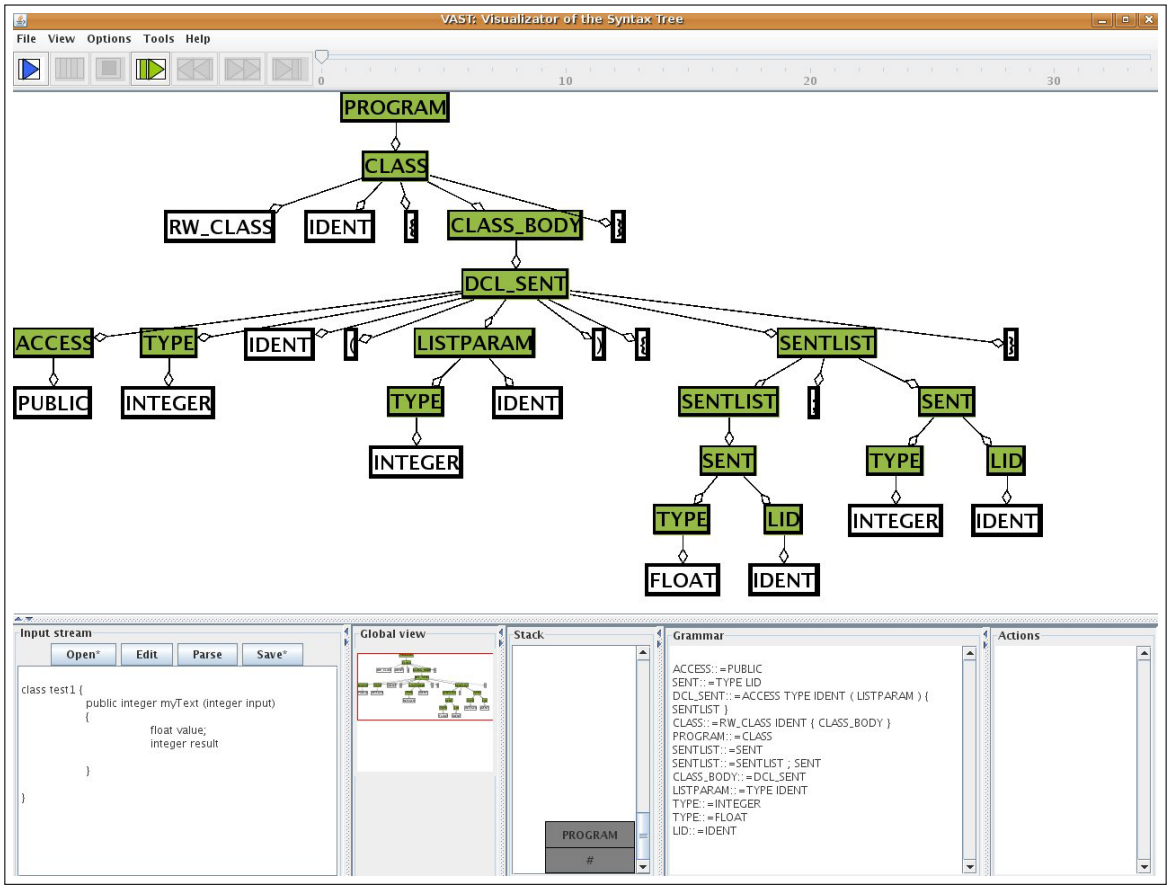


Figure 2.1: VAST main user interface.

The way of working with this version of VAST is divided in three stages: design, execution and visualization. In design step the user has to include manually the calls to VASTapi. Once the lexical and syntax specifications has been modify, the parser has to be generated and compiled using the command line. When the parser is executed, the intermeddle representation should be loaded into VAST in order to be visualized.

2.4.1 Description

In this evaluation participated **59 students** of the *Language Processors* subject of the Rey Juan Carlos University in 2008-2009 course. The participation was voluntary and incentive-based, 2% over the exam mark only if they passed the exam. This evaluation was designed as a controlled experiment plus a observational study. It is a controlled evaluation because students were divided in two groups, treatment group which worked with VAST and the control group which worked with ANTLRWorks. The group's design was performed random but controlled. It was used the marks of a pretest to divided the students, although the assignment to control/treatment group was randomize.

The independent variable was the tool used: VAST (treatment group) or ANTLRWorks (control group). The dependent variable was the educational effectiveness. The students had to document all the tasks done in the evaluation using visualizations and textual explanations. The tasks consisted in 4 exercises about the concepts o the LL(1) analysis (Almeida-Martínez and Urquiza-Fuente, 2009).

2.4.2 Results

Two weeks after the evaluation we performed a postest. According to the Bloom's Taxonomy (Bloom et al., 1959) we found significant differences in the level of *Synthesis* (t(48)=-2.51, **p=0.02**) in favor of the treatment group.

2.4.3 Discussion

The results show that there is significant statistical differences in favor of VAST in the level of *Synthesis*. The general results show that the students have performed worse the postest than the pretest, so the differences are negative. We revised the experiment design and we realized some aspects that could produce these results. On the one hand, there was a period of vacation after the evaluation, which prevent from doing the postest. On the other hand we have to take into account that the pretest was done after finishing the theory of LL(1) parsers, and then lessons continued. The result is that students forgot concepts, although those who used VAST obtained better results.

2.5 Evaluation II

The results of the previous evaluation indicated that the students who used VAST remembered the way of working of LL(1) parsers better that those who used ANTLRWorks. In this case we realized that the way of using VAST was not the most appropriate due to the characteristics of the tool. On the one hand, we thought that the time to use the tool was not enough (2 hours), so the students has to use a new interface and work with the exercises in the same session. On the other hand, we observed that the tasks planned for the evaluations were not the most appropriate. As results, we planned a long term evaluation in which students had to used VAST to design grammars.

2.5.1 Description

In this evaluation participated voluntary **16 students** of the subject *Compilers and Interpreters* of the Rey Juan Carlos University in the 2009-2010 course. As in the previous evaluation we used a pretest to divided the students in two groups; one used VAST and the other the pair of tools JFlex-Cup to solve the exercises. The independent variable was the tool used: VAST for the treatment group and JFlex-Cup for the control group. The dependent variable was the educational effectiveness, according to the differences between postest-pretest. In this evaluation we used 4 sessions (8 hours) to work with VAST.

The tasks performed by the students should be documented at the end of the session using textual explanations and visualizations (only treatment group). The tasks consisted in 3 exercises about the design of grammar for LR(1) parsers.

2.5.2 Results

After finishing the evaluation we analyzed the time invested to study the subject by both groups. We did not find significant differences which indicated that one group were more motivated with the subject after performing the evaluation ($t(9)=3.15$, $p>0.05$). There were significant differences in levels of *Understanding* ($t(9)=0.95$, $\mathbf{p<0.01}$) and *Synthesis* ($t(9)=0.21$, $\mathbf{p=0.03}$) of the Bloom's taxonomy (Bloom et al., 1959). Besides we found significant differences in the global learning $t(9)=0.69$, $\mathbf{p=0.02}$.

2.5.3 Discussion

The results of educational effectiveness obtained in this evaluation show significant differences in two of the levels in the Bloom's taxonomy (Bloom et al., 1959) (understanding and synthesis) and in the global learning of the students. The exercises of the evaluation and the long period of time using VAST have allowed the students, who used VAST, to get more familiar with the tool. The exercises were designed to build grammars and in the level of understanding and synthesis were found significant improvements. The fact of working with VAST during a long period of time and display the syntax tree, according to the grammar and the input stream has allowed to improve the design of grammars.

Although VAST is also design to work with exercises of the application level, in this one we have not found any significant difference because we did not ask the students to work with the input stream processing.

Finally, according to the global learning, we found significant differences in favor of VAST. This was caused, as we have described before, because the long period of working with VAST and the design of the exercises.

2.6 Evaluation III

This evaluation was destined to evaluated the educational effectiveness of the visualization of the syntax error recovery. Before performing this evaluation, we did a preliminary one with other students. From the results of that evaluation we planned a real evaluation of the syntax error recovery.

2.6.1 Description

In this evaluation participated **19 students** of the *Languages Processing* subject of the Rey Juan Carlos University in the 2010-2011 course. The participation was voluntary and incentive-based in 2% over the final mark if the students pass the exam.

This evaluation was designed as an educational effectiveness and observational study. The students were divided in two groups, control and treatment. The control group used the tools JFlex-Cup. The treatment group used VAST. In order to create the groups we used a pretest of knowledge. The students were divided using the pretest but group assignment was random. The pretest included questions according to the different levels of the Bloom's taxonomy (Bloom et al., 1959). The independent variable is the tool used by each group; VAST for the treatment one and JFlex-Cup for control group. The dependent variables were the educational effectiveness according to the learning between a pretest and posttest. This evaluation last one class (2 hours).

2.6.2 Results

The results of the evaluation are divided in educational effectiveness and instructors' observations. The results of educational effectiveness are divided in two parts. On the one hand, the differences between the posttest and pretest. On the other hand, the mark of the exercises done during the evaluation. According to the differences between posttest-pretest, there were not significant differences. In relation with the solutions of the exercises they were analyzed in a global way ($U=45.00$, $p=1.00$) and also by questions independently ($U=36.00$, $p=0.73$). During the evaluation the instructors observed that after 40 minutes all the students of the treatment group had finished the first exercises. At the end of the session, all the students finished all the exercises.

In the control group the instructors observed that the most used visualization was printing the syntax rule used. One student needed to used paper to draw the syntax tree. After 40 minutes, only one student had finished the first exercise. At the end of the session, none of the students had finished all the exercises.

2.6.3 Discussion

The results have been divided in two parts: solutions to the practices and differences between posttest and pretest. The solutions to the practices did not show significant differences. The results obtained in the knowledge test did not show any difference between groups. However, we observed that the students in treatment group (VAST) were able to do the exercises faster than the control group.

2.7 Evaluation IV

The results of the long term evaluation described in 2.5 were satisfactory. From these results we observed that the use VAST during a longer period of time improves the learning task of students. According to this, we decided to perform another long term study formed by three independent evaluations. The objective of this study was to observe if the use of visualization tool motivates the students and help them to learn better.

2.7.1 Description

In this evaluation participated **40 students** of the *Languages Processors* subject of the Rey Juan Carlos University in the 2010-2011 course.

The objective of this analysis was to study the implication of the students in the subject using an optional partial delivery of the obligatory practice of the subject. This delivery was incentive in a 8% over the final mark. To check the practice we used 16 different criteria: solution to grammar ambiguities,

operator precedence, semantic actions, test cases, etc. We distinguished three groups (G0-G2). The *G0* was the group with the students who used ANTLR and JFlex-Cup in the previous evaluations. The *G1* was the group with the students who used VAST. Finally, the *G2* was formed by the students who participated in just one session.

2.7.2 Results

The analysis of the results between the groups G0-G1 (U=22.50,p=0.07), with average marks 7.25 and 11.27 respectively show that the marks were very different. The minimum and maximum marks of G1-G2 oscillate in the range 6-16, while the marks of G1 oscillate in the range 11-16.

Other aspect to consider was the implication of the students. In this delivery participated 24 students. We observed that the 54.27% (13/24) of the students had used VAST in previous evaluations. The 12.5% (6/24) of students had used ANTLR and JFlex-Cup. Finally, the 20.83% (5/24) of the students used a tool in just one occasion.

According to the students who did not participated in the delivery (a total of 16), we observed that the 37.5% (6/16) had used VAST. The 12.5% (2/16) used ANLTR and JFlex-Cup.

2.7.3 Discussion

The results obtained in this analysis of long term allowed to distinguish two fundamental aspects, learning improvement and students engagement.

According to the learning improvement, the marks obtained in the practices show that there is not significant differences between any group. This means that the use of VAST has not influenced in the partial delivery. However, analyzing the marks between control and treatment groups (G0-G1), we observed that the average marks of the treatment group were higher than in control group. Besides, the minimum marks of the students who used VAST were higher than the rest. These results show that VAST can help the students to make better the practices. According to the students implications,

We observed that students who used VAST were more implicated with the subject. In relation to this the number of students who delivered the practice and used VAST was superior than those who used ANTLR and JFlex-Cup.

2.8 Conclusions and future works

The main objective of this work is to maker easier the comprehension of the syntax directed translation (SDT). According to this, it is necessary to understand the previous stage to use correctly de SDT. As the base of the SDT is the syntax stage of the compilation process, it is necessary to get a good representation of this stage.

We have analyzed several tools to display the syntax stage of the compilation process and we have realized that all of them suffer from two limitations: particularity and partiality. We say that a solution is particular because all the tools depends on a specific generator tool. We say that a solution is partial because none of the tool perform a whole visualization of the syntax analysis.

Analyzing the problems of the current tools, we have presented VAST (Visualizer of the Syntax Tree) and its educational use. VAST tries to solve the particularity and the partiality problems using a generic approach. We have performed different educational evaluation of the tool with satisfactory results.

As future works, we plan to improve the compatibility of VAST with more generation tool. On the one hand it is necessary to add more metaparsers to VASTAPÌ in order to make easier the annotation process. On the other hand we will include the compatibility with the EBNF notation from both points of view, visualization and behavior.

Acknowledgment

This project is supported by project TIN2008- 04103/TSI of the Spanish Ministry of Science and Innovation.

Bibliography

Alfred V. Aho. Teaching the compilers course. *ACM SIGCSE Bulletin*, 40(4):6–8, 2008. ISSN 0097-8418. doi: http://doi.acm.org/10.1145/1473195.1473196.

Alexander Aiken. Cool: A portable project for teaching compiler construction. *ACM SIGPLAN Notices*, 31(7):19–24, 1996. ISSN 0362-1340. doi: http://doi.acm.org/10.1145/381841.381847.

Francisco J. Almeida-Martínez and Jaime Urquiza-Fuente. Teaching LL(1) parsers with VAST- An Usability Evaluation-. Technical Report 2009-01, Universidad Rey Juan Carlos, 2009.

Francisco J. Almeida-Martínez and Jaime Urquiza-Fuentes. Syntax trees visualization in language processing courses. In *Proceedings of the Nineth IEEE International Conference on Advanced Learning Technologies, 2009. ICALT 2009.*, page In press, Los Alamitos, USA, 2009. IEEE Computer Society Press.

Francisco J. Almeida-Martínez, Jaime Urquiza-Fuentes, and J.Ángel Velázquez-Iturbide. Visualization of syntax trees for language processing courses. *Journal of Universal Computer Science*, 15(7):1546–1561, 2009.

Francisco J. Almeida-Martńez, Jaime. Urquiza-Fuentes, and J.Ángel. Velázquez-Iturbide. Educational visualizations of syntax error recovery. *EDUCON10: 1st Annual Education Engineering, IEEE*, pages 1019–1027, 2010.

B. Bloom, E. Furst, W. Hill, and D.R. Krathwohl. *Taxonomy of Educational Objetives: Handbook I, The Cognitive Domain*. Addison-Wesley, 1959.

Jean Bovet and Terence Parr. ANTLRWorks: an antlr grammar development environment. *Software: Practice and Experience*, 38(12):1305–1332, 2008. ISSN 0038-0644. doi: http://dx.doi.org/10.1002/spe.v38:12.

R. N. Chanon. Compiler construction in an undergraduate course: some difficulties. *ACM SIGCSE Bulletin*, 7(2):30–32, 1975. ISSN 0097-8418. doi: http://doi.acm.org/10.1145/382205.382886.

Akim Demaille. Making compiler construction projects relevant to core curriculums. In *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 266–270, New York, NY, USA, 2005. ACM. ISBN 1-59593-024-8. doi: http://doi.acm.org/10.1145/1067445.1067518.

William G. Griswold. Teaching software engineering in a compiler project course. *ACM Journal of Educational Resources in Computing*, 2(4):3, 2002. ISSN 1531-4278. doi: http://doi.acm.org/10.1145/949257.949260.

Cris D. Hundhausen, Sarah A. Douglas, and John T. Stasko. A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages and Computing*, 13(3):259–290, 2002.

Andrews Kristy, Henry Robert, and Yamamoto Wayne. Design and implementation of the UW illustrated compiler. In *PLDI '88: Proceedings of the ACM SIGPLAN 1988 conference on Programming Language Design and Implementation*, pages 105–114, New York, NY, USA, 1988. ACM. ISBN 0-89791-269-1. doi: http://doi.acm.org/10.1145/53990.54001.

Henry F. Ledgard. Ten mini-languages: A study of topical issues in programming languages. *ACM Computing Surveys*, 3(3):115–146, 1971. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/356589.356592.

Ronit Ben-Bassat Levy and Mordechai Ben-Ari. We work so hard and they don't use it: acceptance of software tools by teachers. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 246–250, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-610-3. doi: http://doi.acm.org/10.1145/1268784.1268856.

Daniel Resler and Dean Deaver. VCOCO: A Visualisation Tool for Teaching Compilers. In *ITiCSE '98: Proceedings of the 6th annual Conference on the Teaching of Computing and the 3rd annual Conference on Integrating Technology into Computer Science Education*, pages 199–202, New York, NY, USA, 1998. ACM. ISBN 1-58113-000-7. doi: http://doi.acm.org/10.1145/282991.283123.

Susan H. Rodger, Jinghui Lim, and Stephen Reading. Increasing interaction and support in the formal languages and automata theory course. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 58–62, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-610-3. doi: http://doi.acm.org/10.1145/1268784.1268803.

3 Towards the Use of Sequence Diagrams as a Learning Aid

João Paulo Barros*+, Luís Biscaia*, and Miguel Vitória*
*LabSI[2] & ESTIG, Instituto Politécnico de Beja, Rua Pedro Soares, Beja, Portugal
+UNINOVA-CTS, Portugal

joao.barros@ipbeja.pt, lcds.biscaia@gmail.com, migueljvt@hotmail.com

3.1 Introduction

Compared to imperative programing, object-oriented programming brings additional complexities. These complexities are especially challenging for the novice and, as a consequence to the teacher. Hence, it is no surprise that the teaching and learning of object-oriented programming is an extremely popular topic in computer science education research.

This work in progress paper presents the objectives and structure of a tool under development for novice object-oriented programmers that intends to ease code understanding. That is accomplished through the use of sequence diagrams, one of the most popular behavior diagrams in the Unified Modeling Language (OMG, 2011), the *de facto* standard for object-oriented modelling. More specifically, the tool allows the generation of execution traces as sequence diagrams: for a given program run, the student is able to visualize the respective execution as a sequence diagram. Next, we present the Java2Sequence tool.

3.2 The Java2Sequence tool

Object-oriented programming brings additional layers of indirection right from the beginning. More specifically, operations can be executed by classes or by objects and variables can have values or addresses of values. Sequence diagrams offer a simple and visual representation for these indirections, namely the class-object duality, as well as for object creation and method calling. More specifically, the Java2Sequence tool has the following objectives, which when taken together are not, to the best of our knowledge, fulfilled by any available tool:

1. To ease program comprehension for novices learning object-oriented programming using Java;

2. The dynamic generation of UML sequence diagrams, a well-known type of behavior diagram, but enriched with informative extra annotations, while offering a visual representation for several fundamental concepts, namely classes and objects lifelines, object creation, instance and class method calling, parameters, return values, and recursion;

3. A step by step execution of Java code;

4. The generated sequence diagrams should allow (a) filtered views for the simplification and reduction of the resulting diagrams and (b) the visualization of each thread role in program execution.

5. Integration with other tools for teaching programming at the introductory level; we foresee the BlueJ programming environment (Kölling, 2011) and the Violet UML editor (Horstmann and de Pellegrin, 2011); another possibility will be a UMLGraph specification as it supports sequence diagrams (Spinellis, 2008);

Presently, objectives 3, 4b, and 5 are still not completed.

3.2.1 Annotations

The UML specification is quite short regarding annotations for sequence diagrams. Hence, an important addition is the definition of meaningful and informative annotations for each diagram element namely lifelines, activations, and arrows. When these annotations are too long to be presented, they are hidden and replaced by a small plus sign. They are made visible putting the mouse over the plus sign. The examples in the Section 3.2.3 exemplify some of these annotations. Next, we present a brief overview of the tool architecture.

3.2.2 Architecture

The tool is structured in three parts: (1) the core that executes the program being analyzed and handles events from the observed application; (2) the internal representations for the sequence diagram; (3) one or more packages that generate different model specifications. These latter specifications can be textual formats (e.g. XML files), allowing interchange and visualization in other tools, or graphical representations requiring distinct implementations (e.g. Java Swing based diagrams). Fig. 3.1 illustrates the three part decomposition, the relation to the observed application and the resulting specifications.

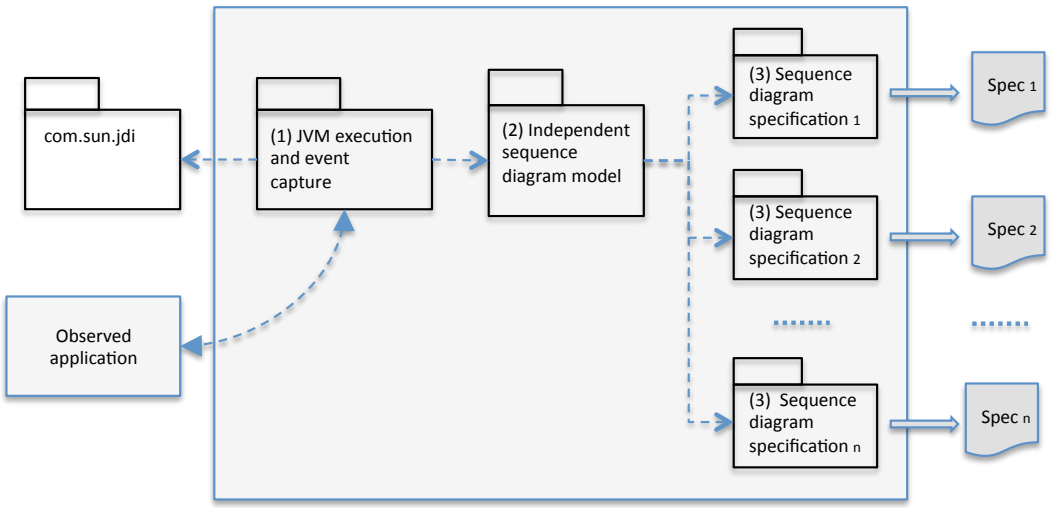


Figure 3.1: Tool structure showing relation with the observed application and generated specifications.

3.2.3 Examples

Here, we present two short examples. The first was more briefly presented in (Barros et al., 2011). It uses a simulator for a vending (dispenser) machine: the `main` method creates a `Dispenser` object, which creates a `MoneyBox` object. Then, the `main` method asks the `Dispenser` object to add a new `Product` object. Finally, the `main` method asks the `Dispenser` object to execute the `insertCoin` operation with an `int` parameter with value `20`. This simple example also illustrates object delegation as the `insertCoin` operation is delegated to the aggregated `MoneyBox` object. It is important to stress that this kind of programs where several objects interact, are especially interesting for sequence diagram, as it becomes possible to visually show the different objects and classes, as well as the respective interactions. Nevertheless, after this example, we present a shorter one, with no objects. The intent is to show that recursion can also be visualized as multiple subactivations of a given instance method.

Listing 3.1: Dispenser machine

```
package ipbeja;

public class Dispenser {
   private List<Product> products; // products in machine
   private MoneyBox moneyBox; // handles money

   public Dispenser()  {
```

```
        this.products = new ArrayList<Product>();
        this.moneyBox = new MoneyBox();
    }

    public void addProduct(Product p) {
        this.products.add(p);
    }

    public int insertCoin(int coin) {
        return this.moneyBox.insertCoin(coin);
    }

    public static void main(String[] args) {
        Dispenser dispenser = new Dispenser();
        dispenser.addProduct(new Product("cookies", 100));
        dispenser.insertCoin(20);
    }
    // ...
}
```

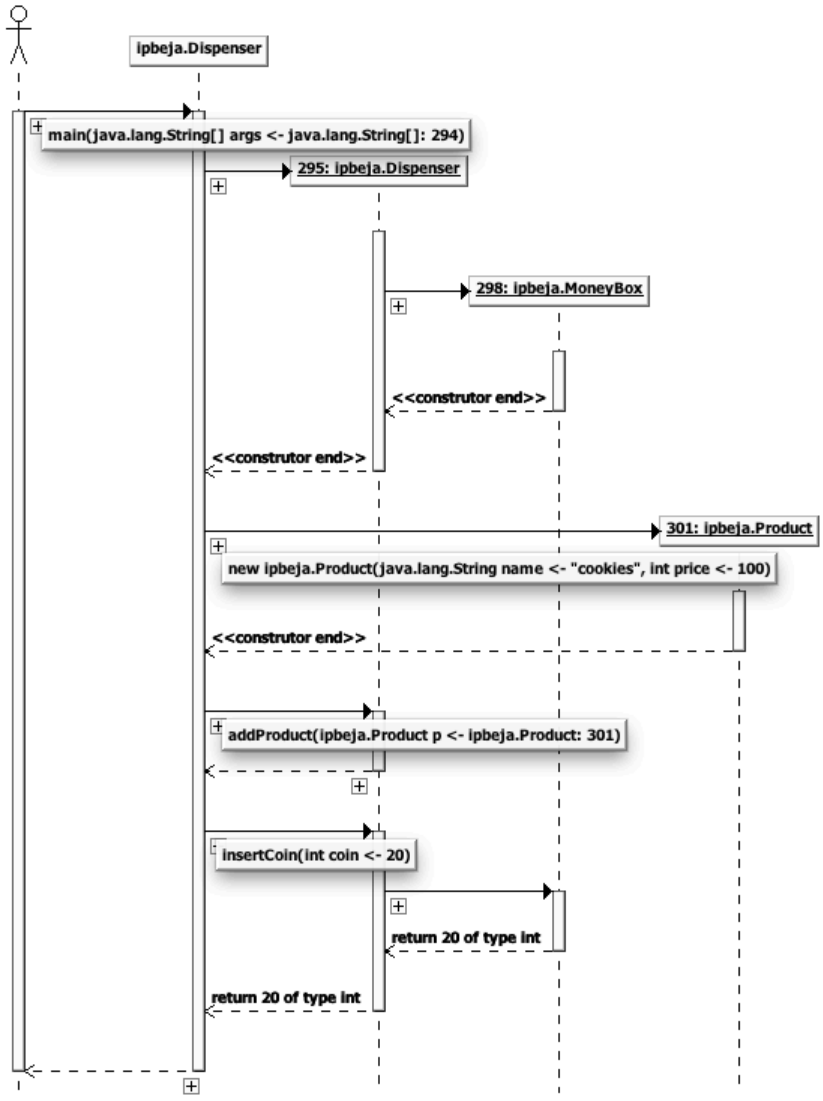


Figure 3.2: Sequence diagram generated from Listing 3.1.

The second example in Listing 3.2 and Fig. 3.3 shows recursive calls for a short Fibonacci computation.

Listing 3.2: Fibonacci calculation

```
package fibonacci;
```

```
public class Fibonacci {

    public static void main(String[] args) {
       Fibonacci s = new Fibonacci();
       s.fib(2);
     }

     public int fib(int n) {
        if(n == 0 || n == 1) {
           return n;
        }
        else {
           return fib(n - 1) + fib(n - 2);
        }
     }
}
```

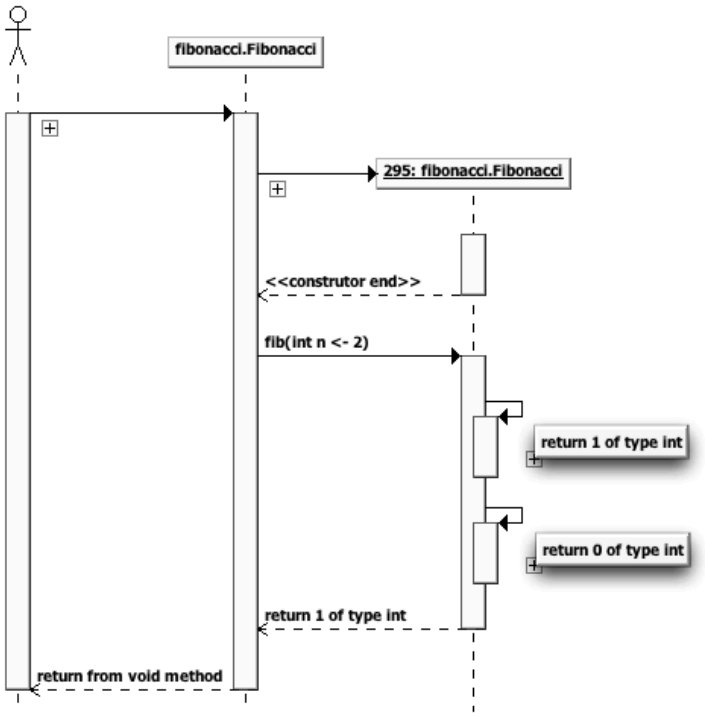


Figure 3.3: Sequence diagram generated from Listing 3.2.

3.3 Related Work

The use of UML in a introductory learning context has been the subject of numerous contributions. Especially due to the size and complexity of the UML specification and even of each of its diagrams, some significant efforts have been made towards the development of simpler and more user friendly UML tools, e.g. (Turner et al., 2005; Ramollari and Dranidis, 2007; Horstmann and de Pellegrin, 2011). Here, we reference related articles and freeware tools more directly related to the generation of UML sequence diagrams from Java source code.

The JACOT tool (Leroux et al., 2003) emphasizes the visualization of thread states using several different diagrams. Some other proposals also emphasize the visual understanding of concurrency, e.g. (Malnati et al., 2008) and (Mehner, 2002). The latter is presented as work in progress and uses the UML CASE Tool Together (from www.togerthersoft.com, now part of Borland.com) for deadlock detection and analysis based on tracing. Differently, regarding concurrency, Java2Sequence emphasizes method calling. Then, for each method call, it should be possible to see the responsible thread.

Oechsle and Schmitt (2002) present the JAVAVIS tool, which uses the Java JDI interface and allows step by step drawing of UML sequence and object diagrams. It also allows the visualization of each

thread role. Yet, it does not foresee integration with other educational tools and, to the best of our knowledge, it is not available.

The JAN tool relies on code tagging to generate sequence diagrams from Java code. As stated: "if program understanding is the objective, carefully planned tagging is required to produce a highly informative animation" (Lohr and Vratislavsky, 2003).

Another type of tool is presented by Grati et al. (2010). It generates a sequence diagram but from a set of executions traces based on use-case scenarios. Then, those execution traces are automatically aligned to produce a combined trace. A general overview of trace exploration tools is presented by Hamou-Lhadj and Lethbridge (2004). Yet, the presented tools do not use UML sequence diagrams from Java source code, although some use similar notations. Briand et al. (2006) provide a methodology to reverse engineer scenario diagrams — partial sequence diagrams for specific use case scenarios — from dynamic analysis in distributed systems in Java/RMI context. The JavaCallTracer tool (Naqvi, 2011) produces sequence diagrams from Java code. Yet, it offers no integrations with other tools and it is targeted to advanced end users. Trace4J (Inghelbrecht, 2009) seems to have similar objectives to our own, but, when tried, it was not available at the specified address (`http://www.trace4j.com`).

Finally, in (Merdes and Dorsch, 2006) the reader can find an interesting discussion, and also some additional references, related to the development of sequence diagram generators from Java source code.

3.4 Future Work

As future work, the tool will expand to fulfill the missing objectives: (1) integration with other tools for teaching programming at the introductory level; (2) support for a step by step execution of Java code; (3) visualization of each thread role in program execution.

Bibliography

João Paulo Barros, Luís Biscaia, and Miguel Vitória. Java2Sequence Ű A Tool for the Visualization of Object-Oriented Programs in Introductory Programming. In *Proceedings of the 16th annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11. ACM, 2011. Accepted poster, to appear.

L.C. Briand, Y. Labiche, and J. Leduc. Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software. *Software Engineering, IEEE Transactions on*, 32(9):642 –663, sept. 2006. ISSN 0098-5589. doi: 10.1109/TSE.2006.96.

H. Grati, H. Sahraoui, and P. Poulin. Extracting Sequence Diagrams from Execution Traces Using Interactive Visualization. In *Reverse Engineering (WCRE), 2010 17th Working Conference on*, pages 87 –96, oct. 2010. doi: 10.1109/WCRE.2010.18.

Abdelwahab Hamou-Lhadj and Timothy C. Lethbridge. A Survey of Trace Exploration Tools and Techniques. In *Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research*, CASCON '04, pages 42–55. IBM Press, 2004. URL `http://portal.acm.org/citation.cfm?id=1034914.1034918`.

Cay S. Horstmann and Alexandre de Pellegrin. Violet UML Editor, 2011. URL `http://alexdp.free.fr/violetumleditor/page.php`. Accessed on 2010/04/28.

Yanic Inghelbrecht. Object-oriented Design with Trace Modeler and Trace4J. In *Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education*, ITiCSE '09, pages 375–375, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-381-5. doi: http://doi.acm.org/10.1145/1562877.1563017. URL `http://doi.acm.org/10.1145/1562877.1563017`.

Michael Kölling. BlueJ – The Interactive Java Environment, 2011. URL `http://www.bluej.org`. Accessed on 2011/04/29.

Hugo Leroux, Christine Mingins, and Annya Réquilé-romanczuk. JACOT: A UML-Based Tool for the Run-Time Inspection of Concurrent Java Programs. In *2nd International Conference on the Principles and Practices of Programming in Java*, 2003.

K.-P. Lohr and A. Vratislavsky. JAN - Java Animation for Program Understanding. In *Human Centric Computing Languages and Environments, 2003. Proceedings. 2003 IEEE Symposium on*, pages 67 – 75, oct. 2003. doi: 10.1109/HCC.2003.1260205.

G. Malnati, C.M. Cuva, and C. Barberis. JThreadSpy: A Tool for Improving the Effectiveness of Concurrent System Teaching and Learning. In *Computer Science and Software Engineering, 2008 International Conference on*, volume 5, pages 549 –552, dec. 2008. doi: 10.1109/CSSE.2008.11.

Katharina Mehner. JaVis: A UML-Based Visualization and Debugging Environment for Concurrent Java Programs. In Stephan Diehl, editor, *Software Visualization*, volume 2269 of *Lecture Notes in Computer Science*, pages 643–646. Springer Berlin / Heidelberg, 2002. URL `http://dx.doi.org/10.1007/3-540-45875-1_13`. 10.1007/3-540-45875-1_13.

Matthias Merdes and Dirk Dorsch. Experiences with the Development of a Reverse Engineering Tool for UML Sequence Diagrams: a Case Study in Modern Java Development. In *Proceedings of the 4th international symposium on Principles and practice of programming in Java*, PPPJ '06, pages 125–134, New York, NY, USA, 2006. ACM. ISBN 3-939352-05-5. doi: http://doi.acm.org/10.1145/1168054.1168072. URL `http://doi.acm.org/10.1145/1168054.1168072`.

Syed Ali Naqvi. Java Call Trace to UML Sequence Diagram, 2011. URL `http://sourceforge.net/projects/javacalltracer/`. Accessed on 2010/04/28.

Rainer Oechsle and Thomas Schmitt. JAVAVIS: Automatic Program Visualization with Object and Sequence Diagrams Using the Java Debug Interface (JDI). In *Revised Lectures on Software Visualization, International Seminar*, pages 176–190, London, UK, 2002. Springer-Verlag. ISBN 3-540-43323-6. URL http://portal.acm.org/citation.cfm?id=647382.724668.

OMG. UML 2.x Superstructure Specification, 2011. URL http://www.omg.org/spec/UML/. Accessed on 2010/04/28.

Ervin Ramollari and Dimitris Dranidis. StudentUML: An Educational Tool Supporting Object-Oriented Analysis and Design. In *in Proceedings of the 11th Panhellenic Conference on Informatics*, 2007.

Diomidis Spinellis. Automated Drawing of UML Diagrams, 2008. URL http://www.umlgraph.org/. Accessed on 2010/04/28.

Scott A. Turner, Manuel A. Pérez-Quiñones, and Stephen H. Edwards. minimUML: A Minimalist Approach to UML Diagramming for Early Computer Science Education. *Journal on Educational Resources in Computing*, 5, December 2005. ISSN 1531-4278. doi: http://doi.acm.org/10.1145/1186639.1186640. URL http://doi.acm.org/10.1145/1186639.1186640.

4 SRec as a Plug-in of BlueJ for Visualizing Recursion

Antonio Pérez-Carrasco, J. Ángel Velázquez-Iturbide
Escuela Técnica Superior de Ingeniería Informática
Universidad Rey Juan Carlos
Móstoles, Madrid, 28933

{antonio.perez.carrasco,angel.velazquez}@urjc.es

4.1 Introduction

Integrated development environments (IDEs) are very important tools for programming education. They support both programming tasks and programming teaching processes. The teacher has available both professional and educational IDEs. Each class of IDEs has its own advantages and disadvantages with respect to education. Professional IDEs support a higher number of functions, but they may be too complex for education. Educational IDEs are simpler and closer to educational activities but they often provide limited functionalities.

Both classes of IDEs often provide extension capabilities so that external libraries or software applications can be integrated, typically as plug-ins. Plug-ins allow educational IDEs to better support teaching and learning, because the user uses the combined functionality of the two software tools into a single software tool. User effort is considerably reduced: the user only has to learn how to use one system, and she does not have either to synchronize both tools or to switch continuously between them. Consequently, both tools are used with higher usability and the instructional goals are achieved with higher effectiveness and efficiency.

In this paper, we present a plug-in of SRec for BlueJ. SRec is an animation system for recursion (Velázquez et al., 2008) and BlueJ is a well-known educational IDE for object-oriented programming. In the second section we introduce the main features of both systems. In the third section we motivate the creation of this plug-in and we describe the use of the plug-in. Finally, we briefly provide our conclusions.

4.2 SRec

SRec is a standalone, educational application for recursion animation. The system visualizes algorithms programmed using the Java language. Currently, it provides three dynamic and synchronized views of recursive algorithms (the activation tree, the control stack and the trace) (Velázquez et al., 2008). It also supports two additional views for divide-and-conquer algorithms (Velázquez et al., 2009) (Velázquez and Pérez, 2008). Finally, it provides a number of educational facilities (internationalization, export/import, etc).

4.2.1 Working with SRec

When a user wants to animate the visualization of a recursive computation, he/she must just load a Java class into SRec, select the method to execute, and introduce the parameters of the execution. In the case of a divide-and-conquer algorithm, he/she must also select the methods for which the divide-and-conquer views will be activated. Then, SRec launches the execution of the selected algorithm. During the execution, SRec gathers all the information it needs to display the recursive process. Once the execution is finished, SRec creates the different views in its main window (see Fig. 4.1), and lets the user animate the execution (using the collected information) in the activated views.

SRec includes a number of educational facilities. Thus, SRec is internationalized to support multiple languages (currently, English and Spanish). SRec also allows exporting the contents of the views to standard graphical formats (PNG, GIF, JPG). This support is useful for both teachers and students as it makes easy documenting reports, slides or web pages. SRec also allows saving the animation in a single animated GIF file (notice that animated GIF format does not support interaction).

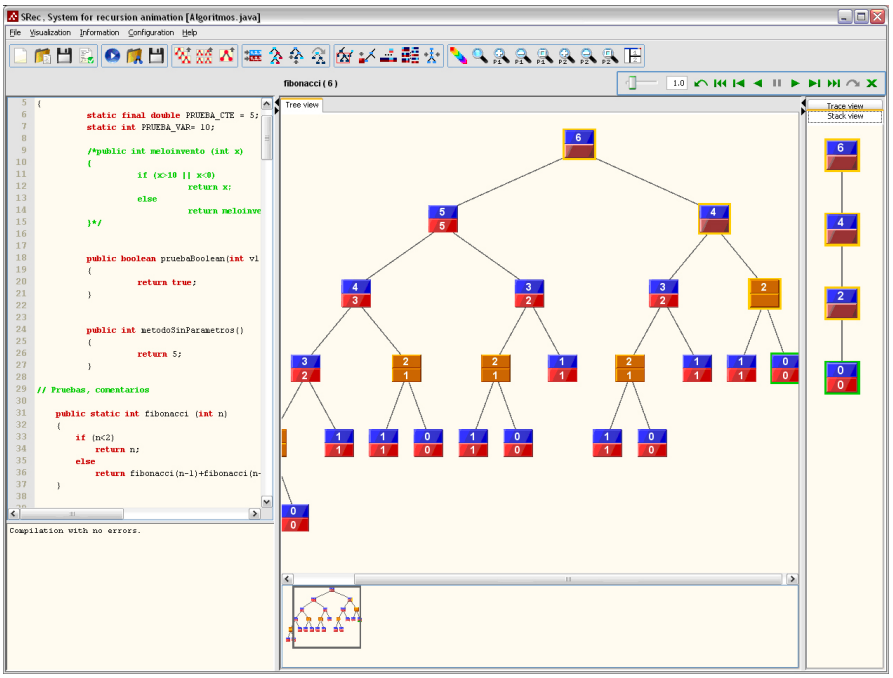


Figure 4.1: Main window of SRec

Finally, the user may save an animation so that it may be loaded and played again in the future. This way, it is not necessary to go through the creation process leading to the visualization. Consequently, the teachers can create a library of algorithm animations ready to use, thus saving lecture time. The students can also save their visualizations to submit them to the teacher, and ask questions about the execution of the corresponding algorithm.

4.2.2 Interaction Functionality

SRec offers a number of classes of interaction with the visualizations which allow users to comprehend and analyze algorithm behavior more flexibly. Firstly, the animation bar allows inspecting the execution both manually and automatically, forwards and backwards. In any of these directions, the user can control an animation step-by-step (i.e. recursive steps), advance to its end, and jump over a particular recursive call; the latter possibility allows users to directly obtain a partial result without the inner details.

Secondly, the user can customize a high number of typographic features of fonts, colors, frames and distances.

Thirdly, the user can modify the amount and format of the information that SRec displays in the views. SRec allows the user to show/hide some data like input values or output values. The user also may show/hide the nodes corresponding to methods he/she is not interested in (for example, to ignore a combination operation in a divide-and-conquer algorithm). Past recursive calls are held in the recursion tree view but they may also be attenuated or hidden.

Fourthly, SRec provides three different interactions to handle large recursion trees. SRec allows scrolling the tree view, so that he/she can easily navigate to examine any part of the tree. There also are several zoom functions, which allow enlarging or making smaller nodes and their contents. Notice that the larger number of nodes is displayed, the fewer nodes fit in the window, and vice versa. Finally, SRec provides an overview+detail interface for the recursion tree where user can simultaneously observe the whole tree with minimum size in a frame and a part of the tree at a larger size and in a second frame. The user can click or move the frame for selecting the part of the tree he/she wants to view in more detail.

Finally, SRec offers miscellanea of other interaction functions. Thus, he/she may highlight nodes with the same contents (so that the user may analyze redundancy in calculations, see Fig. 4.1), rearrange the layout of the visual items in some views, and obtain statistical summaries about either the active node or the whole computation. For the former statistical summary, SRec gives information about the number of descendant nodes of the active node, the state of the node (pending, finished, etc.), the value of all its parameters and results, or the complete identifier of the function it represents. For the latter statistical

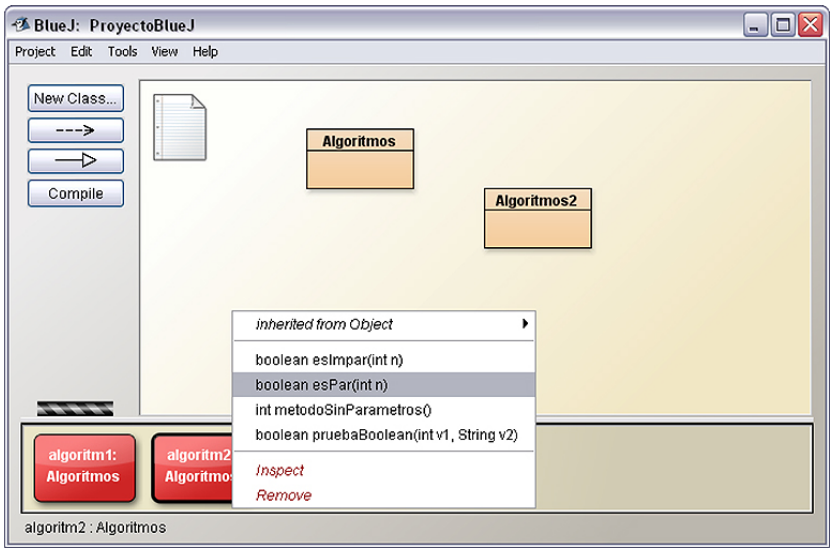


Figure 4.2: Main window of BlueJ

summary, the user may know the total number of recursive calls that the algorithm invoked, number of recursive calls displayed so far, number of recursive calls which were skipped, number of finished recursive calls, which methods are involved in the whole execution, etc.

4.3 BlueJ

BlueJ (Kölling et al., 2003) is an educational IDE aimes at introducing object-oriented programming. It uses the Java language. BlueJ provides a very simple, intuitive and visual interface that makes easy editing the Java code of classes and the execution of object methods.

BlueJ was designed following some guidelines about visualization, interaction and simplicity. For example, code editing is made through an editor that colors the different elements of Java syntax (keywords, identifiers, numbers, operators, brackets...) and the domain of classes, methods, flow control structures, etc. for helping user to find syntax errors.

BlueJ uses UML diagrams for representing the set of classes that is being handled by the user at each moment. This UML diagram shows the relations between classes (inheritance, instantiation...), and allows the user to keep in his mind the global structure of the program he/she is programming, so he/she can easily understand and control the relations between classes and between objects created from these classes. The main window is shown in Fig. 4.2.

In BlueJ, objects can be created without writing code for the constructor methods. This way, the tool demands less effort to debug and to analyze object-oriented programs, an untypical feature in programming environments.

4.3.1 Working with BlueJ

After the user launches the application and opens a project, an UML diagram is drawn in the main window that shows the available classes and their relations. The user can interact with them instantaneously for creating objects, editing their code, executing a method, adding new classes to the program, launching messages on objects and reviewing their state.

Objects are represented as red boxes located in the lower zone. BlueJ allows the user to inspect the contents (i.e. the state) of each object to check how it changes along the user launches messages on them. Everytime the user invokes a static or dynamic method, he/she has to provide valid values for the input parameters.

According to BlueJ authors, students usually need less than 30 minutes to use the application deeply without any problem, so it is ideal for university context, where students always have some problems with the lack of time for learning new tools. In fact, some students are reluctant to leave the program for adopting more complex environments that have more interesting features and possibilities because they feel very comfortable working with BlueJ.

4.3.2 BlueJ extensions

BlueJ offers an API to software developers for making easy the capture of events like the compilation of classes, the edition of one class code, the execution of a method, etc. These captures make very easy the synchronization between BlueJ and an external application connected as a plug-in.

A number of plug-ins for BlueJ can be found at the tool web page (Kölling, 2011). They have extended BlueJ functionality, applicability and dissemination. Currently, BlueJ is used in over a thousand universities and colleges around the world.

4.4 The Plug-in

The creation of a plug-in is very interesting for developers and for students: the former have available opportunities for their applications and the latter get available integrated functionality that improves their user experience. Our plug-in integrates a programming IDE, very well accepted by the university community, with an application that provides the capacity of visualizing and interacting with recursive algorithms. This extends BlueJ capabilities for detailed analysis of recursive algorithms. In this section, we explore other plug-ins and present the motivation for creating the plug-in and its technical features.

4.4.1 Related work

Currently, there are a number of plug-ins for IDEs that generate some benefits with respect to standalone systems. Eclipse is a professional integrated development environment used in companies around the world. This environment is very extended in the university community too. Eclipse is a professional IDE but there are available many educational plug-ins. For instance, AnimalScript (Röβling and Schroeder, 2008) allows the users edit script code for Animal (Röβling and Freisleben, 2002), a tool for creation, modification and presentation of visualizations and animations of algorithms. Another example of Eclipse plug-in (Lintern et al., 2003) supports SHriMP, which is a stand-alone application for visualizing program artefacts (documentation, code, classes, relations, etc). Consequently, the new plug-in better helps students in understanding their own programs.

There also are many plug-ins for BlueJ, which give support for programming robots, formatting texts, checking format of programming code or interpretation of a Javascript code ins. For instance, there is a plug-in (Paterson et al., 2006) that allows the user to choose a design pattern using a wizard. The user can customize the name of the classes, and the new generated code is integrated into the current project. This makes using design patters easier, so better code is obtained.

We want to remark the existence of a Jeliot plug-in (N. Myller et al., 2007) for BlueJ because it is very similar to the SRec plug-in. Jeliot is an "standalone" application for introducing programming using Java (Moreno et al., 2004). It provides, through a metaphorical theater, the capacity of playing the execution of an algorithm. Operations such as assignments, expression calculations, calls to methods or the behaviour of objects can be visualized step by step and so students may understand how they work. Everytime a user launchs a method in BlueJ, Jeliot opens a new visualization that allows analyzing step by step the algorithm. As a consequence, the user obtains extended functionality (i.e. visualization with Jeliot) without additional effort.

Research on educational software can also be supported by BlueJ by means of another plug-in (Jadud, 2006) (Jadud and Henriksen, 2009) that allows the user to monitor BlueJ actions. The plug-in allows knowing how novel programmers learn to program (how many code editions he/she does before compiling, how long it takes him/her to recompile the code, etc). This plug-in works because it has access to an API provided by BlueJ that captures events like compilation or code edition. The plug-in connects to a database for storing gathered data that can be later analyzed by researchers.

4.4.2 Motivation for Creating this Plug-in

The creation of a plug-in generates benefits for the two parts: applications and their developers, and users. In particular, the benefit that a BlueJ plug-in provides to SRec is that SRec is made visible and available to a wider audience (teachers and students). BlueJ users will know about SRec when they visit the BlueJ web page searching for plug-ins, libraries or other resources for that environment. The plug-in also is beneficial for BlueJ because it can offer to its users a new functionality, not provided so far, to visualize and analyze recursive computations.

Figure 4.3: Dialog box for activating or deactivating the plug-in in BlueJ

Users who wished to visualize recursion will not have to switch from BlueJ, where they edit Java code, to SRec, where recursion is visualized. So far, they had to develop their programs with BlueJ and then to load the Java class into SRec. Another choice was to develop the code in SRec, whose editor is useful but not as elaborated as BlueJ's. This second option requires user effort to learn the use of the SRec editor.

4.4.3 Installation

The installation of the SRec plug-in is very easy. The user just needs to download the ZIP file from the plug-in web page (Pérez, 2011). Then, he/she must uncompress it at the "/lib/extensions" folder located within the installation folder of BlueJ. As a consequence, the IDE will find the plug-in automatically when it is launched.

The ZIP file contains the executable file of SRec as a plug-in. Notice that this executable file does not work as a standalone application or as a plug-in for another IDE. The ZIP file contains three folders for configuration files and classes required for compilations in runtime. After the file has been uncompressed and correctly located, BlueJ will load it automatically when it starts, so user has not to make any configuration action.

The user can activate and disable the plug-in at any time of BlueJ usage. He/she only has to select "Activate/Deactivate SRec" at the "Tools" menu. After that, BlueJ launches a dialog box, showed in Fig. 4.3, where user can choose the desired option: activate or deactivate SRec.

4.4.4 Using the Plug-in

When the plug-in is already installed, it is very simple to use. Every time BlueJ runs a method of a class, SRec runs it too in order to gather information for its visualization. If the plug-in window is not visible, it will appear automatically. Every time BlueJ executes a method, SRec substitutes the previous visualization with the new one. The SRec window can be closed by the user anytime. When the user wants to stop using BlueJ and the SRec plug-in, he/she just has to close the BlueJ window, and the window of SRec is also closed.

State diagrams of BlueJ with SRec plug-in are shown in Fig. 4.4.

Interaction facilities with the visualization are the same in the plug-in and in the standalone version of SRec, so the user can make the same kinds of work with both versions. The plug-in and the standalone version allow the user to select which data he/she wants to visualize, to export views to graphical formats, etc. The only difference with this first version of the plug-in is that it does not support the activation of divide-and-conquer views.

Some options about loading and editing classes also are disabled because the behaviour of the plug-in is mastered by BlueJ and the classes it handles. BlueJ reports SRec which class it has to load, which method it has to run, and the value of the parameters. The only way to watch Java code is by editing the class in BlueJ, as shown in Fig. 4.5.

SRec is not aimed at visualizing object-oriented algorithms. When the user tries to visualize algorithms that create or handle objects with SRec, he/she can find a runtime error. The teaching of algorithms is often made in procedural programming, so the plug-in can be widely used without finding some errors.

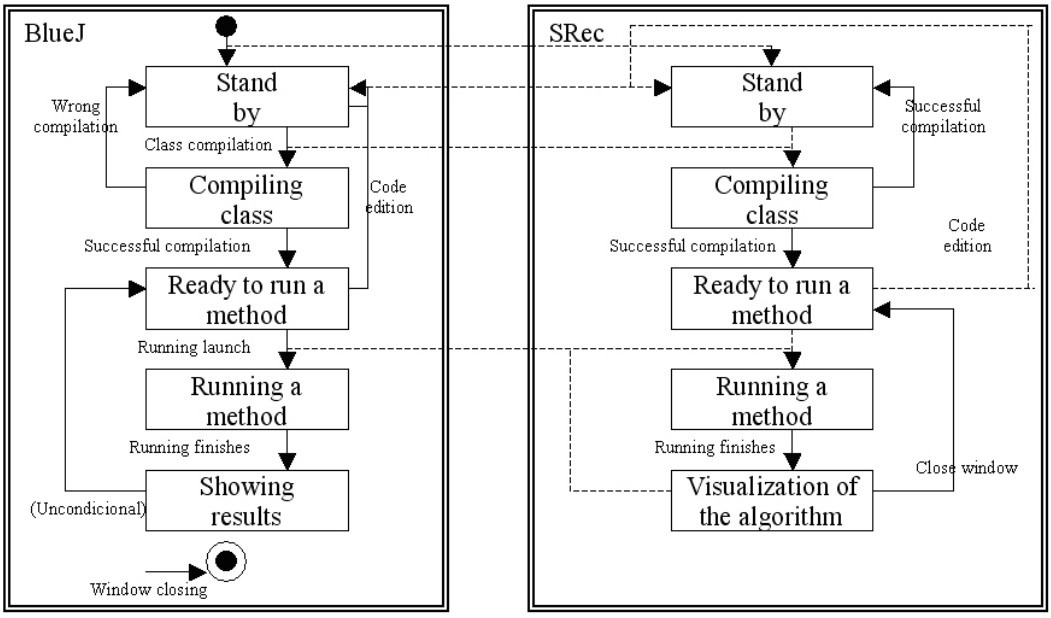


Figure 4.4: States diagram for BlueJ and the plug-in of SRec

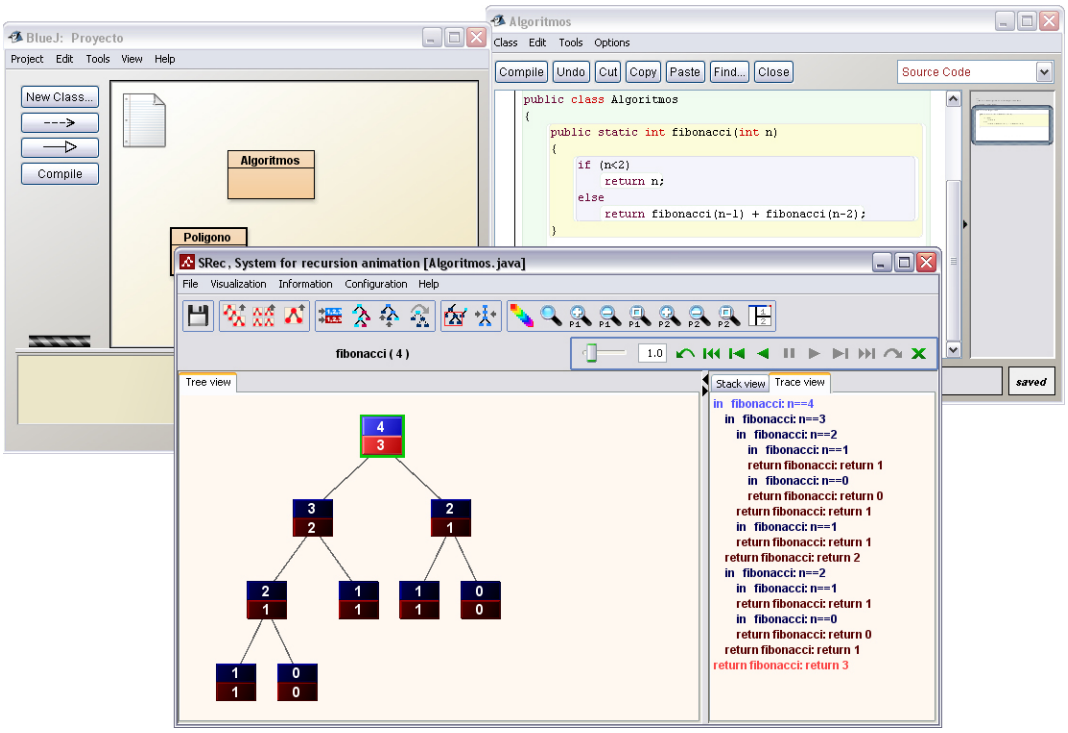


Figure 4.5: Working with the plug-in of SRec and the code editor of BlueJ

A plug-in of SRec for BlueJ has been introduced. SRec is an educational tool for helping students to analyze recursive algorithms. It is a BlueJ plug-in, a programming environment aimed at teaching the object-oriented paradigm using the Java language (SRec is not object-oriented, so that feature is not needed).

The main features and technical aspects of SRec and BlueJ have been reviewed. They have merged their capabilities, allowing users editing code, running methods and visualizing recursive processes with just one tool. It makes user interaction easier and probably enhances user effectiveness and efficiency.

We have some plans for the near future. On the one hand, we would like to develop more plug-ins of SRec for others environment like Eclipse or NetBeans. On the other hand, we would like to make a study on whether the SRec plug-in really improves efficiency and increases the satisfaction of users. For this work, we can use log files generated by SRec, which gather information about how long a task takes to students or how many errors they produce while they are using SRec.

Acknowledgments. This work is supported by research grant TIN2008-04103/TSI of the Spanish Ministry of Science and Innovation.

Bibliography

M.C. Jadud. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the second international workshop on Computing education research (ICER 2006)*, pages 73–84, 2006.

M.C. Jadud and P. Henriksen. Flexible, reusable tools for studying novice programmers. In *Proceedings of the fifth international workshop on Computing education research workshop (ICER 2009)*, pages 37–42, 2009.

M. Kölling. Extensions for bluej, April 2011. URL `http://www.bluej.org/extensions/extensions.html`.

M. Kölling, Bruce Quig, Andrew Patterson, and John Rosenberg. The BlueJ system and its pedagogy. *Journal of Computer Science Education*, 13(4):249–268, 2003.

R. Lintern, J. Michaud, and M-A. Storeyand X. Wu. Plugging-in visualization: experiences integrating a visualization tool with eclipse. In *Proceedings of the 2003 ACM symposium on Software visualization (SOFTVIS 2003)*, pages 47–56, 2003.

A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari. Visualizing programs with Jeliot 3. In *Proceedings of the working conference on Advanced visual interfaces (AVI 2004)*, pages 373–376, 2004.

R. N. Myller, Bednarik, and A. Moreno. Integrating dynamic program visualization into BlueJ: the Jeliot 3 extension. In *Proceedings of Seventh IEEE International Conference on Advanced Learning Technologies (ICALT 2007)*, pages 505–506, 2007.

J.H. Paterson, J. Haddow, and M. Nairn. A design patterns extension for the BlueJ IDE. In *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education (ITICSE 2006)*, pages 280–284, 2006.

A. Pérez. Plug-in of SRec for BlueJ, April 2011. URL `http://www.lite.etsii.urjc.es/srec/bluej/`.

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

G. Röβling and P. Schroeder. Animalipse: An Eclipse plug-in for AnimalScript. In *Proceedings of Fifth Program Visualization Workshop (PVW 2008)*, pages 97–104, 2008.

J.Á. Velázquez and A. Pérez. SRec 1.2: visualizador integrado de programas recursivos generales y de divide y vencerás. In *Proceedings of XI International Symposium on Computers in Education (SIIE 2009)*, 2008.

J.Á. Velázquez, A. Pérez, and J. Urquiza. SRec: An animator system of recursion for algorithm courses. In *Proceedings of 13th Annual Conference on Innovation and Technology in Computer Science Education (ITICSE 2008)*, pages 225–229, 2008.

J.Á. Velázquez, A. Pérez, and J. Urquiza. A design of automatic visualizations for divide-and-conquer algorithms. *Electronic Notes in Theoretical Computer Science*, 224(1):159–167, 2009.

5 Characterizing Time and Interaction in a Space of Software Visualizations

J. Ángel Velázquez-Iturbide
Dpto. Lenguajes y Sistemas Informáticos I, Escuela Técnica Superior de Ingeniería Informática
Universidad Rey Juan Carlos
Móstoles, Madrid, 28933

angel.velazquez@urjc.es

5.1 Introduction

Software visualization is a technology that raised much expectation in past decades for programming education. Research efforts were initially focused on specification and implementation techniques, but that the use of visualizations did not always result in educational enhancement was quickly made clear.

Analyses of educational evaluations have led to the conclusion that more important than the visualizations themselves is the educational use we make of them (Hundhausen et al., 2002). Based this premise, a taxonomy with levels of student engagement has been proposed (Naps et al., 2003), so that the higher student engagement with a visualization, the higher educational impact. The taxonomy is in common use as a framework, and has led to numerous evaluations of usage with positive results (Urquiza and Velázquez, 2009). In this context, it is important to further study the role of human-computer interaction in educational software visualizations.

This paper is a step towards the ultimate goal of characterizing interaction in software visualization systems. In particular, we define the space of software visualizations and analyze the different representations of time. The paper is structured as follows. The second section contains a brief overview of interaction in visualizations. The third section deals with the space of software visualizations and the representations of time. In the fourth section we briefly illustrate our proposal by characterizing interaction in two very different visualization systems. In the fifth section we study the integration of our proposal into a comprehensive taxonomy of software visualization systems. Finally, we include our conclusions.

5.2 Related Work

Information visualization is a subfield of human-computer interaction that Card et al. (1999) define as: "the use of computer-supported, interactive, visual representations of data to amplify our cognition". The use of computers and their visual nature are obvious features of information visualizations, but notice the emphasis on their interactive nature.

A number of taxonomies have been proposed in the past to characterize interaction in information visualization. Our use in this paper of a particular taxonomy is as an instrument to discuss and illustrate the specific features of interaction in software visualization. Consequently, we adopt the taxonomy by Yi et al. (2007) because of its high-level nature. In particular, it classifies the different kinds of human-computer interaction based on the user's intent, that is, "what the user wants to achieve". This concept is quite effective to classify the low-level interaction techniques into a small number of high-level categories. In the sequel, the authors propose seven interaction categories: select, explore, reconfigure, encode, abstract/elaborate, filter, and connect. We briefly introduce each category by quoting a sentence from Yi et al. (2007) that summarizes its meaning:

- Select. Select interaction techniques provide users with the ability to mark a data item of interest to keep track of it.
- Explore. Explore interaction techniques enable users to examine a different subset of data cases.
- Reconfigure. Reconfigure interaction techniques provide users with different perspectives onto the data set by changing the spatial arrangement of representations.

- Encode. Encode interaction techniques enable users to alter the fundamental visual representation of the data including visual appearance (e.g., color, size, and shape).
- Abstract/ellaborate. Abstract/elaborate interaction techniques provide users with the ability to change the level of abstraction of a data representation.
- Filter. Filter interaction techniques enable users to change the set of data items being presented based on some specific conditions.
- Connect.Connect refers to interaction techniques that are used to (1) highlight associations and relationships between data items that are already represented, and (2) show hidden data items that are relevant to a specified item.

Software visualization is the subfield of human-computer interaction that refers to the visualization of computer software. Price et al. (1998) define it as follows: "the use of the crafts of typography, graphic design, animation, and modern cinematography with modern human-computer interaction and computer graphics technology to facilitate both the human understanding and effective use of computer software".

Interaction also plays an outstanding role in the latter definition, but it actually is very limited in most software visualizations, being almost exclusively limited to animation. We may check this claim by taking a look at the treatment given to interaction in taxonomies of software visualization. Several taxonomies do not consider any kind of interaction but animation. The taxonomies by Roman and Cox (1993) and by Price et al. (1998) are the only ones that give a relevant role to interaction, but not as central as in information visualization taxonomies. Both taxonomies identify several categories to describe a software visualization system. Roman and Cox (1993) indirectly define the category "interface" with the question: "what facilities does the system provide for the visual display of information?" Interaction is a subcategory that gathers the tools that the user may use to modify the display. They consider two interaction forms: by means of controls, and by means of the visualization. Notice that both forms are low-level and limited.

Price et al. (1998) indirectly define the interaction category with the question: "how does the user of the software visualization system interact with and control it?" They further identify three subcategories: style, navigation, and scripting facilities. The first subcategory is similar to the interaction subcategory by Roman and Cox, and the third one refers to export/import facilities. The second subcategory considers elision control and temporal control, but they fail to consider other kinds of interaction.

In summary, information visualization strongly emphasizes human-computer interaction with visualizations, and a large number of different interactions have been identified. However, software visualization only considers a small number of interactions, being animation the most relevant.

5.3 Characterization of Interaction in Software Visualization Systems

In past years we developed a number of software visualization systems for different domains. We characterized them, and for this task we have sought appropriate taxonomies. For instance, we used Yi et al.'s taxonomy to characterize interaction in the SRec animation system for recursion (Velázquez and Pérez, 2010). However, we found several difficulties and currently we consider that no taxonomy exists to appropriately characterize interaction in software visualization systems.

The difficulty we face here is the lack of an adequate treatment of time. Information visualizations deal with very complex sets of data, that typically are static. Time plays an outstanding role in software visualizations in the form of animations. However, time deserves a more sophisticated treatment. For instance, animations may not only illustrate the lineal-time behavior of an algorithm execution over a particular data set, but of several running algorithms. In addition, time itself may sometimes be represented within static visualizations, as an alternative to animation.

In this subsection we deal with time and other issues, which are prerequisites for addressing more satisfactorily interaction with software visualizations.

5.3.1 Interacting with Time

An animation is the dynamic display of a graphical representation of software execution. A user typically interacts with software visualizations in two ways: over a static visualization (representing a state of its execution) or over the animation (to make a transition between two execution states). The former interaction acts over space whereas the latter acts over time. However, the representation of time is not

always restricted to animations. For instance, the user may be shown a sorted sequence of visualizations representing software states. In this case, time is implicitly represented by the sequence of visualizations, which is spatially available to the user.

According to Gómez (1995), time can be represented in four ways:

- Animation. The system provides the user a set of functions to control animation. These functions typically mimic videotape recorder controls: rewind, step backwards, automatic run, step forwards, forward (up to the end), etc. The user is usually able to adjust the animation speed.
- Small multiples. A sequence of consecutive static visualizations is displayed.
- 3D. The program state is represented in a plane and time is used as the third dimension where to display the successive states.
- 1/2D. Time is embedded into the visualization by juxtaposing the current state and some representation of past states. An instance of this kind of representation is recursion trees, where the current and past recursive calls are drawn.

Notice that the three last representations of time make use of space. In these cases, interaction with past instants of the execution is reduced to interaction with their spatial representation. However, interaction with invisible states or with an animation must be done over time.

5.3.2 Interaction Interfaces

According to the analysis made above, the user may interact with software visualization through different means. She may use the animation controls to navigate in time and she may also interact with the active static visualization which represents the current program state. Furthermore, it could be that she is also able to interact with past states through the active static visualization.

We speak of an interaction interface to refer to a part of the user interface of a software visualization system that allows the user interacting with time or space in a specific way. An interaction can be implemented with different low-level techniques: direct manipulation, menu option, mouse click, etc. In general, a software visualization system has several interaction interfaces. Thus it is common that the animation controls provide an interaction interface and the static visualization offers a second one.

We may further elaborate the features of an interaction interface by considering that it may support three kinds of interactions:

- Space. It supports interaction with a static visualization that represents the current program state.
- Space/time. It supports interaction with a static visualization that represents both the current program state and information of past states.
- Time. It does not provide a spatial representation of time, but interaction is provided by means of animation.

If we want to characterize the interaction supported by a software visualization system, we must specify all of its interaction interfaces. Each interaction interface can be specified by means of an interaction table. An interaction table is a bidimensional table with one or two columns, depending on the kind of interaction it supports:

- Space. It has one column (labeled “space”) describing space interactions with the static visualization.
- Space/time. It has two columns (labeled “space” and “time”), where one describes the space interactions with the static visualization and the other describes its time interactions.
- Time. It has one column (labeled “time”) describing time interactions with the animation.

In Section 4, we characterize several interaction interfaces of two visualization systems by means of interaction tables. In the present section we still have to make one additional point clear.

5.3.3 A Space of Software Visualizations

If we analyze the literature on software visualization, we notice that the situation is more complex than just interacting with the visualization of one program execution. We find the following, complex visualization scenarios:

- Multiple views (e.g. JCAT (Brown and Raisamo, 1997)). Several graphical representations of the same piece of software under execution are simultaneously displayed in coordination. Each view typically highlights a specific issue, therefore the different views are simpler and more understandable than a single, comprehensive view.
- Multiple algorithms (e.g. SortingOutSorting (Baecker, 1998)). The simultaneous execution of several algorithms is displayed. It is used to compare the behavior of the different algorithms, often remarking their performance differences.
- Multiple input data (e.g. HalVis (Hansen et al., 2002)). An algorithm is illustrated with several animations, each one showing the algorithm behavior for different input data.

Given all of these possibilities and also the different forms of visualizing the execution of a single algorithm, we may think of a space of software visualizations. In general, a software visualization system may handle static visualizations in a space of four dimensions:

- Algorithm.
- View.
- Input data.
- Execution time.

This space may be represented as a hypercube. A point in this space represents a specific view of an execution state of a particular algorithm for given input data. A line, a plane or a hyperplane also make sense in this space of software visualizations. For instance, the typical animation of an algorithm (with a specific view and for given input data) corresponds to a line along the execution time axis.

Our final remark is that an interaction interface supports user interaction with a subset of the space of software visualizations. Thus, if interaction is limited to a static visualization, the interface allows the user interacting with the corresponding point in the space. An interaction interface supporting animation allows the user interacting with the corresponding line along the execution time axis. Interaction interfaces that give access to planes or the hyperplane also are possible, as we will show in the next section.

5.4 Examples of Characterizing Interaction

We illustrate the concepts introduced in the previous section by characterizing two software visualization systems. We have chosen two systems developed by our research group because of familiarity and also because they provide very different interaction facilities. In order to be concrete, we use the taxonomy by Yi et al. (2007) cited in Section 2, which considers seven categories of interaction based on the user's intent. Notice, however, that other interaction taxonomies could be used. For a different interaction taxonomy, the table would have the same structure but in the number of rows (equal to the number of interaction categories in the taxonomy) and the row titles (the names of the corresponding categories).

5.4.1 Interaction in the SRec System

SRec (Velázquez et al., 2008) is a system aimed at animating the execution of recursive algorithms. Fig. 5.1 shows its user interface with two views: the recursion tree at the central panel and the control stack at the right panel. Notice also the animation bar at the top right corner, which animates the different views in coordination.

The animation bar defines an interaction interface over time along the execution time axis. The control stack defines an interaction interface over space for the point representing the current execution state.

Table 5.1: Interaction table for recursion trees in SRec

	Space	Time
Select	-	Search nodes by parameters/results
Explore	Zoom Scrolling bar Interface overview+detail	Make a past node active
Reconfigure	-	-
Encode	Node color Width and type of node frame Width and type of arrow Vertical gap	Past nodes visible/attenuated/invisible
Abstr./elab.	-	Number of different types of nodes
Filter	Input/output, parameters, methods Show all the values in a node	Recursion subtree
Connect	-	-

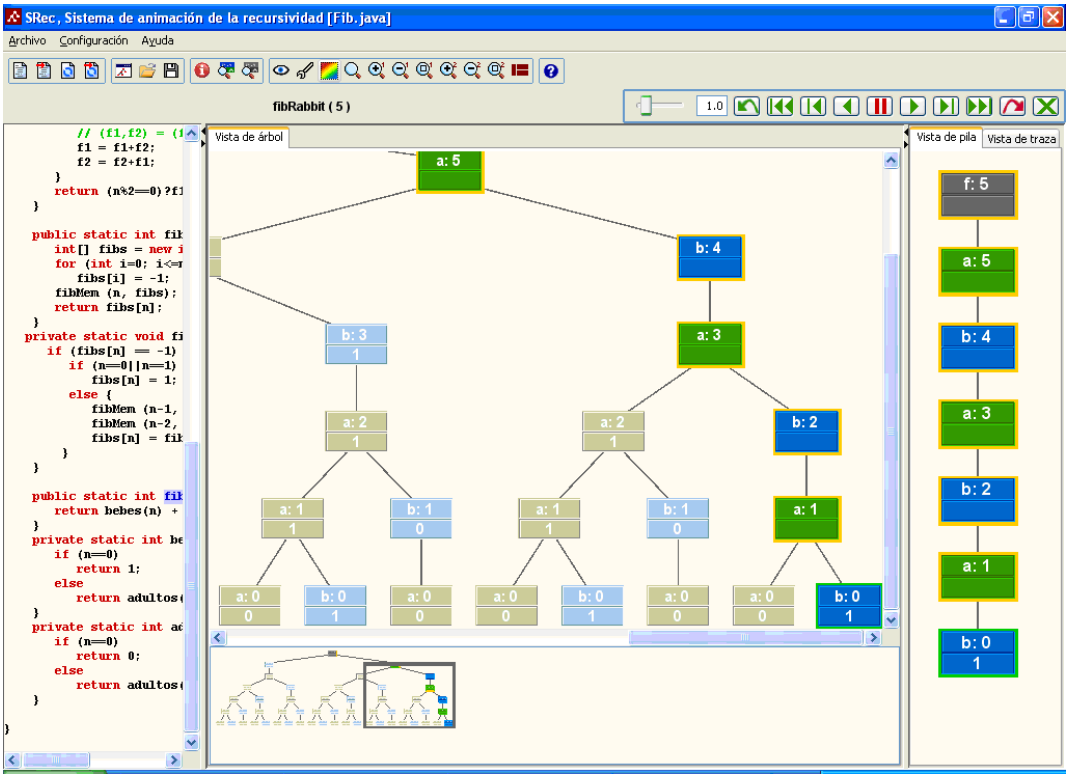


Figure 5.1: User interface of SRec

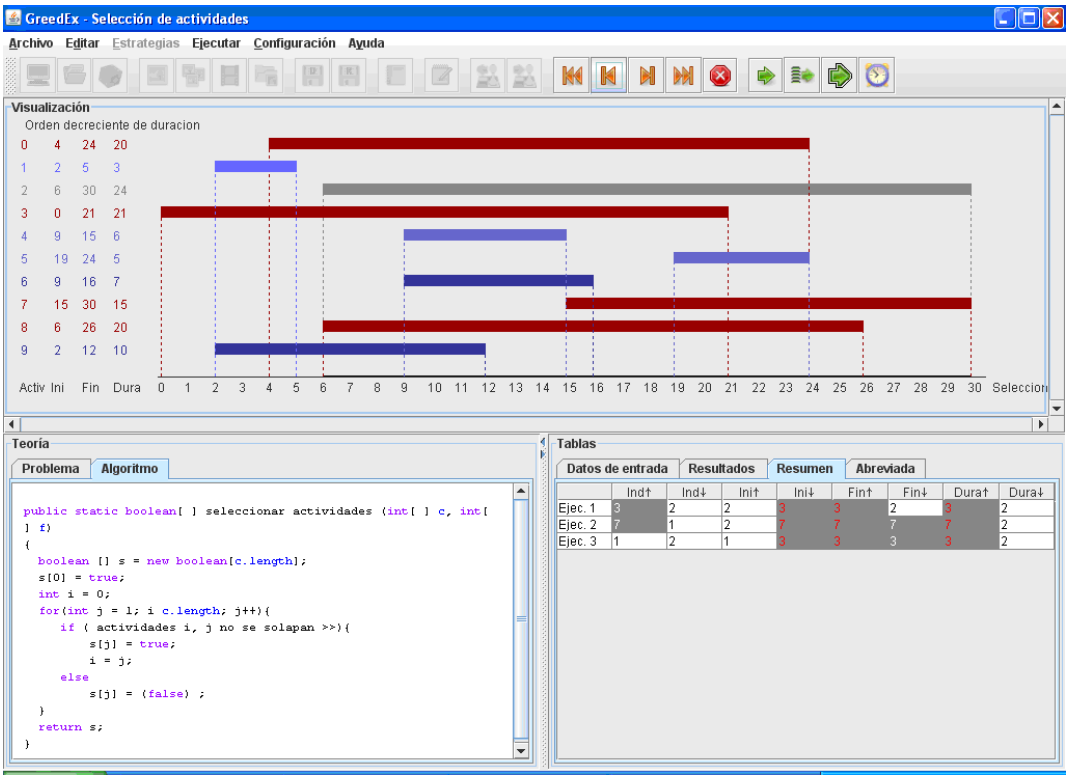


Figure 5.2: User interface of GreedEx

The recursion tree is more interesting as it allows interacting with time (past states) and with space (the current state), as Table 5.1 shows. Notice that the interactions defined on the "space"column allow interacting with the tree spatial representation as a whole. In addition, the interactions defined on the "time"column allow interacting with the spatial representation of past nodes.

5.4.2 Interaction in the GreedEx System

GreedEx (Velázquez and Pérez, 2009) is an interactive assistant aimed at learning the foundations of greedy algorithms. Fig. 5.2 shows its user interface with two views: the visualization panel at the center of the window and the summary table at the bottom right panel. The animation controls are embedded within the icons bar.

The visualization panel and the animation bar of GreedEx allow interacting with space and time in standard ways. The right bottom panel contains four tabs which provide more interesting interactions. In particular, the summary table (see Fig. 5.2) gives access to a plane of algorithm visualizations, each one associated to specific input data and a specific algorithm. This interface supports access (i.e. a "select" interaction) to individual animations. We do not formalize these interaction interfaces as tables because of lack of space.

5.5 Integration into Taxonomies of Software Visualization Systems

Our proposal only deals with some of the features of a visualization system. In order to completely characterize a system, our proposal should be integrated into a general taxonomy of software visualization systems. The taxonomy by Price et al. (1998) is probably the most comprehensive, so we refer to it in this section.

We should treat separately the space of visualizations and interaction. These authors propose six top-level categories, where categories "C: Form" and "E: Interaction" are the most adequate for our purposes.

"Form" may be characterized as "what are the characteristics of the output of the system (the visualization)?" In particular, subcategories C.4 and C.5 are "multiple views" and "program synchronization" (i.e. multiple algorithms), respectively. Our proposal integrates these two subcategories and more. Therefore,

it would be more adequate to join both subcategories into a more general subcategory named "C.4: Space of visualizations". This subcategory can be defined as "What kind of simultaneous visualizations does the system support?" According to the hierarchical nature of the taxonomy, "multiple views", "multiple algorithms", "multiple input data" are relevant sub-subcategories. However, it would be more accurate to identify the interaction interfaces of a system and then to describe the space of visualizations each interface supports.

"Interaction" may be characterized as "how does the user of the software visualization system interact with and control it?" According to Price et al. (1998), the most important interaction subcategory is "E.2: Navigation", but we have seen above that other kinds of interaction also are recommended for some tasks. These kinds of interaction should be incorporated into Price et al.'s taxonomy as new interaction subcategories. Furthermore, each interaction subcategory could be more accurately characterized by using an interaction table, as shown above. We do not advocate here any particular classification of interaction, and our use of Yi et al.'s taxonomy is for illustrative purposes.

5.6 Conclusions and Future Work

Software visualization has specific features that make classifications of interaction in visualization information inadequate to fully characterize it. Its most important, specific feature is the dominant role of time. This short paper is a step towards characterizing interaction in software visualization systems. We have defined the space of software visualizations and analyzed the different representations of time. Both issues have led to the concept of interaction interface, which was specified by means of interaction tables. The proposal was successfully applied to characterize interaction in the SRec and GreedEx systems. Given the very different features of both systems, the generality of our proposal has been illustrated.

We plan to continue refining our proposal. Although the taxonomies by Yi et al. and others allow modelling interaction in information visualization, their application to software visualization is more problematic. Therefore, we must review and restate interaction categories to make them more adequate for characterizing software visualization.

Acknowledgments. This work was supported by research grant TIN2008-04103/TSI of the Spanish Ministry of Science and Innovation.

Bibliography

R. Baecker. Sorting out sorting: A case study of software visualization for teaching computer science. In J.T. Stasko, J. Domingue, M.H. Brown, and B.A. Price, editors, *Software Visualization*, pages 369–381. The MIT Press, Cambridge, MA, 1998.

M.H. Brown and R. Raisamo. Collaborative active textbooks using java. *Computer Networks and ISDN Systems*, 29:1577–1586, 1997.

S.K. Card, J.D. Mackinlay, and B. Shneiderman. *Readings in Information Visualization.* Morgan Kaufmann, San Francisco, CA, 1999.

L. Gómez. *Classification and Use of Concurrent Program Visualization Techniques.* PhD thesis, Technical University of Madrid, 1995.

S. Hansen, D. Schrimpsher, and N.H. Narayanan. Designing educationally effective algorithm animations. *Journal of Visual Languages and Computing*, 13:291–317, 2002.

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

T. Naps, G. Roessling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, and J.Á. Velázquez. Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin*, 35(2):131–152, 2003.

B. Price, R. Baecker, and I. Small. Introduction to software visualization. In J.T. Stasko, J. Domingue, M.H. Brown, and B.A. Price, editors, *Software Visualization*, pages 3–27. The MIT Press, Cambridge, MA, 1998.

G.C. Roman and K.C. Cox. A taxonomy of program visualization systems. *Computer*, 26(12):11–24, 1993.

J. Urquiza and J.Á. Velázquez. A survey of successful evaluations of program visualization and algorithm animation systems. *ACM Transactions on Computing Education*, 9(2), 2009.

J.Á. Velázquez and A. Pérez. Active learning of greedy algorithms by means of interactive experimentation. In *Proceedings of 14th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2009)*, pages 119–223. ACM Press, New York, 2009.

J.Á. Velázquez and A. Pérez. Infovis interaction techniques in animation of recursive programs. *Algorithms*, 3:76–91, 2010.

J.Á. Velázquez, A. Pérez, and J. Urquiza. SRec: An animation system of recursion for algorithm courses. In *Proceedings of 13rd Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2008)*, pages 225–229. ACM Press, New York, 2008.

J.S. Yi, Y.A. Kang, and J.T. Staskoand J.A. Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):224–231, 2007.

6 Increasing the use of JFLAP in Courses

S. H. Rodger, H. Qin, J. Su
Computer Science Department
Duke University
Durham, NC 27708

rodger@cs.duke.edu

6.1 Introduction

Automata theory courses have traditionally been taught with pencil and paper problem solving, resulting in small, tedious to solve problems that are likely to contain errors. In the past twenty years, a number of software tools have been developed. Most tools focus on a particular concept or a set of related concepts, while other tools focus on a wider variety of concepts. We list a few such tools (Barwise and Etchemendy, 1993; Cogliati et al., 2005; Taylor, 1998) that allow users to visualize and interact with concepts from this course.

We have developed JFLAP(Rodger and Finley, 2006; Rodger, 2011), an instructional tool for visualizing and interacting with many concepts in automata theory and formal languages including regular languages, context-free languages, and recursively enumerable languages. With JFLAP one can build an automaton and then step through a simulation on input strings. JFLAP focuses on several types of parsing including brute-force parsing, LL(1) parsing, SLR(1) parsing and CYK parsing. The SLR(1) parsing method involves building a DFA that models the parsing stack, building the parse table, and then parsing strings and stepping through the creation of parse trees. JFLAP also includes interaction with construction-type proofs, such as converting an NFA to a DFA to a minimal state DFA.

We have worked to increase the usability of JFLAP in courses in three ways, 1) by making JFLAP apply to a broad range of topics and general definitions of those topics, 2) by focusing on the interaction and usability of the tool by both students and instructors, and 3) by showing how JFLAP helps students solve problems that are too difficult to solve on paper.

6.2 Usability by range of problems and generality

We have defined items in JFLAP generally to fit with many automata theory textbooks, and have options for limiting the generality so a student can focus on a particular definition. JFLAP has several variations of the finite automaton (FA) including deterministic (DFA), nondeterministic (NFA), and other variants such as Moore and Mealy machines. In JFLAP the label on a transition for an FA has zero, one or multiple symbols. JFLAP will fit with books that have a simpler definition of length zero or one, and with books that allow multiple symbols. Labels in JFLAP now allow a regular expression of the form [m-p] to represent one of the characters in the range from m to p. This allows users to make more realistic FA easily such as an FA to recognize integers (using [0-9] and [1-9] on transitions) or an FA to recognize valid variable names using [a-zA-Z] on transitions.

The pushdown automaton (PDA) transition in JFLAP is defined to allow multiple symbols to read, pop or push. When creating a new PDA, JFLAP prompts for the transition definition to use, either all fields multiple characters (very general) or all fields a single character. By selecting single character mode, that forces the PDA to read exactly zero or one symbol from the tape, to push exactly zero or one symbol onto the stack and to pop exactly zero or one symbol from the stack. This is a standard definition for a nondeterministic PDA (NPDA) that many textbooks use. Later a textbook may generalize fields, such as allowing to push multiple symbols onto the stack. There is one more choice for a PDA, the type of acceptance. When running the PDA on an input string select either accept by final state or accept by empty stack. Note that the NPDA starts with a bottom of stack marker. This marker and everything else must be off the stack to accept by empty stack.

The Turing machine is defined in several ways including one-tape, multi-tape and building blocks. In the definition of the one-tape Turing machine the user reads one symbol from the tape, writes one symbol

to the tape and then moves the tape head left (L), right (R), or stays put (S). Before creating a Turing machine the user can now set several options to narrow the definition. Those options include allowing or disallowing transitions from final states, allowing or disallowing the stay option as a move, accepting by final state and accepting by halting.

Turing machine Building blocks allow a user to create a Turing machine, give it a name and then use the name as a building block within another Turing machine. This makes it easier to create larger and more interesting Turing machines. Building blocks can be connected to regular states or other building blocks. Generalized transitions allow for easier connections. For example, the transition labeled $a; a, S$ (meaning read an "a", write the same symbol "a" and do not move the tape head), can be shorted to a. Even more general, the transition $\sim$ means, read any symbol, write the same symbol and do not move the tape head.

In JFLAP one can also start with a grammar. After entering in the grammar and before parsing the grammar there is an option for listing the type of grammar. This is helpful to students to verify the type of grammar when an instructor has asked them to write a particular type of grammar. The types of grammars checked for include regular grammar (right-linear or left-linear), context-free grammar, context-sensitive grammar, unrestricted grammar, and Chomsky Normal Form (CNF).

Other preferences for generalization to fit with more textbooks include selecting the empty string as either ϵ or λ, and forcing the trap state to appear or making the trap state invisible. The trap state can make some automata very messy looking so the default is to not show the trap state, but some books require that a DFA include an arc out of every state for every symbol in the alphabet.

6.3 Usability by Students and Instructors

Students could use JFLAP in a number of problem solving ways. These problems may be homework problems, problems recreated from class that were worked during class by the instructor, or problems created by students in order to get additional practice with learning a concept or algorithm.

A usability study on JFLAP was run(Rodger et al., 2009) with feedback from students from several universities that showed several strong results in the usability of JFLAP and its use outside the classroom. For example, over 60% of the respondants thought the use of JFLAP in the course made learning course concepts somewhat easier or much easier.

We list several features of JFLAP to increase its usability by students.

- JFLAP has an editor for building a transition diagram for many of the different types of automata. Previously the user had no control of the location of transitions. JFLAP would attempt to best connect them between two states. We have increased the flexibility of the transitions by allowing the user to curve the transitions, and for loop transitions on a single state, to move them to another location on the edge of that state.

- The image in the JFLAP editor pane can now be saved to an image file in a number of formats: png, jpg, gif or bmp, or export to svg format. This feature will be used to include JFLAP pictures in textbooks and papers, for figures in class notes and by students for pictures for homework.

- The layout of a transition diagram is a graph layout problem. This problem is studied in many graph drawing tools and even an annual conference called Graph Drawing. In JFLAP the user places states where they want them and can then move them around. JFLAP includes about a dozen different graph layout algorithms for the user to try to find one that looks right for their automaton. The user can save a layout so they can come back to it if they do not like the results from other layouts.

- There is now an undo and redo capability that stores and revives editing commands, a request from many users.

Instructors use JFLAP in many ways. They use it to prepare slides for class, and to prepare homework problems or problems to work on in class or lab. For some problems they may create an example that students would either further develop or debug. Some instructors use JFLAP during lecture to illustrate new concepts or proofs.

For instructors to use JFLAP during a lecture or in presentations, we have recently modified the tool to include a zoom functionality. For example, in the automata editor pane, there is now a slider bar that can be used to increase the size of the automaton. Similar zooming slider bars appear in the grammar pane and many of the parts of more complex windows that are part of the parsing or construction proofs.

6.4 Usability by Complexity of problems

We give examples to show how one can explore more complex problems with JFLAP.

6.4.1 Example: Universal Turing machine

Using JFLAP we created a Universal Turing machine with just over 30 states that is available for download. It is a 3-tape Turing machine in which tape 1 is the encoding of a Turing machine M, tape 2 represents the current tape of M starting with the encoding of its input and tape 3 holds the current state M is in. A recent change in JFLAP makes it easier to run the Universal Turing machine. JFLAP now allows an input string to be loaded from a file and the input for the Universal Turing machine can be quite large.

6.4.2 Example: parsing equivalent, yet different grammars

This example illustrates the simulation of two equivalent grammars in order to compare them. The grammar on the left has a λ-rule. The grammar on the right is the equivalent grammar after removing the λ-rule and adding rules needed to represent the missing rules.

G1:	$S \to aB$	$B \to b$		G2:	$S \to aB$	$S \to a$	$B \to B$
	$B \to BB$	$B \to \lambda$			$B \to BB$	$B \to b$	$B \to aa$
	$B \to aBa$				$B \to aBa$		

Running the brute-force parser in JFLAP on the left grammar for the string *aabbab* generates over 1800 steps and takes about 8 seconds. Running the same string on the grammar on the right is much faster, about 40 steps and less than a second. The difference between the two grammars is impressive to students when demonstrating the runs during a lecture.

6.4.3 Conversion of DFA to regular expression

This example illustrates that a proof conversion can be too complex and tedious to enter in all the steps and the interaction with the user needs to be reduced to a more reasonable amount. The algorithm JFLAP uses for converting a DFA to a regular expression removes one state at a time until the DFA has just two states and the regular expression is easily obtainable. This algorithm starts by creating an equivalent DFA with only one final state, and creating arcs between all combinations of states (adding $\emptyset$ if there is no transition). Next the user selects one state at a time to remove, picking any state except the initial state or final state. For the state removed, every remaining arc must incorporate information from the removed state in the form of a regular expression. With regular expressions on arcs we now call the transition diagram of the DFA a Generalized Transition Graph (GTG). This part was very complex to create as an interactive tool. We limited the interaction, because it was too tedious for the user to enter in each regular expression. However, we wanted the user to be able to focus in on any regular expression generated. When a user selects a state to remove, JFLAP lists all the transitions not involving that state in a table with the new regular expression that will become the new label for that transition. The user can click on any transition in the table and the transitions from the DFA that form this new transition are highlighted. Once the user has seen enough, the table disappears, the state is removed and the GTG updated with the new regular expression labels. This step is repeated until there are just two remaining states, and it is easy to compute the regular expression.

6.4.4 Comparison between NPDA and SLR Parsing

Our last example shows how multiple parts of JFLAP can be used to study a new concept. We focus on SLR(1) parsing. With JFLAP one can convert a CFG into an NPDA that models the SLR(1) parsing process. One can run the NPDA with input strings and observe the nondeterminism. The user has to guide the parsing by examining the lookahead. In another approach, the same grammar can be converted to an SLR(1) parse table. In this case the SLR(1) parsing is deterministic. It automatically uses the lookaheads since that information is contained in the SLR(1) parse table. Showing the processing of both the NPDA and the SLR(1) parser for the same grammar and string helps reinforce the idea behind the parsing that SLR(1) is about an NPDA that uses lookaheads to choose the next rule.

6.5 Conclusions

We have developed JFLAP to cover a wide variety of topics in the area of automata theory. We have shown how we have made JFLAP more widely usable by making the definitions of its parts general, yet also allow restrictions on the general definitions. We have made changes to JFLAP to make its interface easier to use by students and have made changes that make it easier for instructors to use in the classroom. Finally we have given several examples that show the advantages of using JFLAP to illustrate more complicated examples.

6.6 Acknowledgements

The work of the authors was supported in part by the National Science Foundation through NSF grants DUE-0442513 and DUE-1044191.

Bibliography

J. Barwise and J. Etchemendy. *Turing's World 3.0 for the Macintosh*. CSLI, Cambridge University Press, 1993.

J. Cogliati, F. Goosey, M. Grinder, B. Pascoe, R. Ross, and C. Williams. Realizing the promise of visualization in the theory of computing. *JERIC*, 5:1–17, 2005.

S. H. Rodger. Jflap web site, 2011. `www.jflap.org`.

S. H. Rodger and Thomas W. Finley. *JFLAP - An Interactive Formal Languages and Automata Package*. Jones and Bartlett, Sudbury, MA, 2006.

Susan H. Rodger, Eric Wiebe, Kyung Min lee, Chris Morgan, Kareem Omar, and Jonathan Su. Increasing engagement in automata theory with jflap. In *Fourtieth SIGCSE Technical Symposium on Computer Science Education*, pages 403–407. SIGCSE, March 2009.

R. Taylor. *Models of Computation and Formal Languages*. Oxford University Press, New York, 1998.

7 Perspectives on Algorithm Visualization on Mobile Devices

Ville Karavirta
Aalto University, School of Science
Department of Computer Science and Engineering
ville.karavirta@aalto.fi

7.1 Introduction

Back when the web was young, there was great excitement of the possibilities it offered for computer science education, with Boroni et al. (1998) calling it a paradigm shift. That excitement has been followed by the development of numerous web-based tools and systems to support CS students and teachers. For a couple of years, we have seen a new paradigm shift; this time towards the mobile devices accessing the web. However, the CS education community has not been as keen to develop supporting tools as it has been with the web.

Algorithm Visualization (AV) is one of the fields where web-based educational systems have seen significant research interest as well as a multitude of different systems. The nature of AVs is that they are short, concise presentations of a certain algorithm that take relatively little time to use and learn[1]. Thus, they would be well suited for mobile content.

Like many new technological innovations, mobile device (and mobile learning) has many different definitions. In this paper, we will use a simple definition of a mobile device being a small device that can be used to browse the web (including smartphones and tablets but excluding laptops). A recent review found mobile technologies to be mature enough for prime time use (Lam et al., 2010). Indeed, there have been successful experiments with mobile such as Abilene Christian University giving iPhones/iPod Touches to their first year students[2].

The increase in mobile devices and mobile data consumption has been so rapid that statistics are bound to be outdated soon. According to ComScore[3], in 2010, *"nearly 47 percent of mobile subscribers in the U.S. were mobile media users (browsed the mobile web, accessed applications, downloaded content or accessed the mobile Internet via SMS)"* with a 7.6 percentage points growth from 2009.

For AV to be part of this growth, the goal of this position paper is to motivate the AV community to pursue development activities for the mobile web. Furthermore, this paper positions some of the existing knowledge of AV development and usage to the world of mobile devices and provides initial guidelines and identifies future research questions for AV on mobile devices.

The claims of this paper are: 1) mobile could be the future for AV, 2) smartphones and tablets should be supported first, 3) web applications should be the mobile medium of choice, and 4) mobile devices enable new engagement methods for AV. Sections 2 to 5 will justify these claims. Finally, Section 7.6 will introduce future research challenges.

7.2 Mobile Could to Be the Future for AV

The main challenge of bringing AV to the mobile era is that according to Shaffer et al. (2010) *"Virtually all AVs and AV toolkits... since the mid-1990's were implemented in Java"*. The choice of Java has been clearly the most popular, mainly because it fulfills the requirement of being platform independent. However, the problem is that Java applets and WebStart applications do not work on any of the existing mobile devices.

The multitude of different existing mobile platforms makes selecting a target medium and devices difficult. It is impossible to predict the winning devices in the future in such a young and turbulent

[1] See, for example, (Malmi and Korhonen, 2004) for time usage statistics.

[2] http://www.acu.edu/promise/innovative/mlreport2009-10.html (visited July 5, 2011)

[3] comScore, Inc.: The comScore 2010 Mobile Year in Review. Available online at http://www.comscore.com/Press_Events/Presentations_Whitepapers/2011/2010_Mobile_Year_in_Review (visited July 5, 2011).

market. And as Java was seen as a solution in the desktop world, there is no obvious choice in the mobile world (Section 7.4 returns to this issue with some answers). There is even more fragmentation of devices with varying capabilities than in the desktop world.

The benefits of mLearning mentioned in literature are that it offers flexibility, creates new opportunities to traditional classroom, and enhances quality of work (Seong, 2006). It can be seen as an extension of eLearning with the complete freedom of location and time.

When considering algorithm visualizations, the content is well suited for mobile learning. As mentioned already, the visualizations often concentrate on a single topic and take relatively little time to complete. Thus, they could well be used while commuting to school or home, which were mobile learning scenarios identified by Gil-Rodríguez and Rebaque-Rivas (2010). Furthermore, according to Sharma and Kitchens (2004), mLearning emphasizes more voice, graphics, and animation based instruction than traditional eLearning. Graphics and animation are the backbone of algorithm visualizations as well.

For these reasons, as well as the potential for innovative uses of AV, I strongly believe the future of AV to be limited if no systems are available for mobile learning.

7.3 Support Smartphones and Tablets First

Often the mobile phones are divided into feature phones and smartphones (for example, in Fling (2009)). Feature phones introduced usages such as listening to music, taking photos and accessing the Internet on the phone. What smartphones add to this is larger screens, QWERTY keyboard or stylus input, and high-speed internet access. The latest devices have introduced touch screens and application markets.

According to comScore, the smartphone share in US has risen dramatically (from 2009 16.8% to 27% in 2010) and many EU countries are even ahead of those numbers. Thus, it makes sense for AV community to first focus efforts for these devices instead of trying to support old devices with limited capabilities. This choice is also supported by the increased use of tablet devices (such as iPad) which typically use the same operating systems and applications as smartphones. Furthermore, when considering algorithm visualization content, it makes no sense to try to introduce AV on old phones with small screens and poor browsers. Learning something from a 128x160 pixel[4] screen through visualizations does not sound like such an appealing approach.

7.4 Web Application is the Medium We Should Choose

7.4.1 Mobile Mediums

There are several possible mobile mediums (that is, the ways to publish applications for mobile devices) to choose from. A good overview is provided by Fling (2009). The list below is a shortened list of the options with some mediums combined since differentiating them makes little sense in AV context.

If the requirement is to support as wide an audience as possible, *SMS* (or MMS, Multimedia Messaging Service) is the way to go. While I'm not really suggesting to use SMS for this, it is mentioned here for the sake of completeness. Furthermore, it would be interesting to have an AV "system" with input as an SMS and an MMS visualization as a response.

Mobile websites are a medium where the web page content is optimized for mobile devices. Furthermore, the content is mostly static with little interaction from the user's part. Again, this medium is supported by a wide range of devices that come with mediocre web browsers. Still, it is not terribly interesting from the perspective of engaging algorithm visualizations.

Mobile web applications are web *applications* that are optimized for mobile devices. The difference with mobile websites is that these are typically highly interactive applications mimicking the interactions of desktop software rather than websites. The development of mobile web applications is done using open web technologies such as HTML(5), CSS(3), and JavaScript. The most significant advantage of mobile web applications is the ability to use the same content for both mobile and desktop world. Obviously, this requires work on adapting the content for different screen sizes, but still the effort required to do that is lower than maintaining multiple systems for the different environments.

Native Mobile Platforms are platforms like iOS (operating system of Apple iPhone and iPad), Android, Symbian, Meego, Windows Mobile, RIM, just to mention a few. The obvious downside is the different technologies used to develop applications for these platforms such as Objective-C for iOS and Java for

[4] This screen size is, for example, in Nokia 6111 from late 2005.

Android. This makes it necessary to port applications when wanting to support multiple platforms. Another downside is that native applications are often distributed by a controlled application store. Thus, the applications need to be accepted to the store. An example of a not so successful acceptance procedure is the Scratch application on iPhone which was pulled from the store since no executable code can be downloaded by the application[5].

At the moment, I see three major reasons why building a native application seems lucrative. Native applications make it easy to achieve the same look-and-feel as other applications on the platform. Furthermore, charging for applications and content is made easy by the application stores (not so relevant for research projects, though). Finally, native apps enable the use of services of the platform such as location, camera, accelerometers, or data stored on device (e.g. contact, photos, email). Yet, the reasons for web applications are even stronger.

7.4.2 Reasoning for Web Applications

Simply put, the web application approach is the only choice where developers do not need to go through operators or application stores to get the content on the devices. Furthermore, it is the only choice where the same application can be used on multiple platforms and also in desktop browsers. These are important reasons for the AV research community that has limited resources. Thus, the choice is obvious. There is also other data that supports this choice. According to comScore, 61% of EU5[6] phones have full HTML browsers (46% in US) while the share of mobile consumers who used the browser and who used an application are almost even.

7.4.3 Open Technologies for Web Applications

Selecting web applications as the base of building mobile algorithm visualizations is solving the issue only half way. For building interactive algorithm visualizations, mobile web applications provide generally three approaches to displaying graphics in browsers:

- HTML5 Canvas: The `canvas` is a new element and JavaScript API introduced as part of HTML5. It offers a way to draw bitmap graphics in browsers. As the graphics is pixels instead of objects, it makes manipulation and animation of them more difficult. In addition, it means that there is no support for DOM events (that is, events like mouse clicks or mouse over/out) on the graphics. While this can be a good thing from performance perspective if there are lots of objects, it generally requires the JavaScript to keep track of the objects anyway. Browser support is good with all latest major browsers (excluding IE8) support it along with most newer mobile browsers.

- SVG (Scalable Vector Graphics): SVG enables using vector graphics in browsers. The SVG graphical elements (such as rect, circle) are DOM elements as well and can respond to DOM events such as mouse clicks. There are also good JavaScript libraries for working with SVG (most notably Raphaël[7]). SVG is supported in all major (desktop) browsers (IE9 will support) and many mobile browsers. However, the implementations are not complete (for example, support for animation is somewhat lacking). Notably, SVG is not supported on Android, but will be in a future release[8]. It needs to be mentioned that SVG has been used as an export format for AV in MatrixPro back in 2004 when SVG was a new and "hot" technology (Karavirta et al., 2004)

- CSS3 (Cascading Style Sheets): While CSS3 is not really intended for displaying graphics (in the same sense as canvas or SVG), it does have such versatile properties for styling HTML elements that it can be used for algorithm visualization[9]. New CSS modules have introduced good support for transformations, animation, and even 3D transformations. Furthermore, using CSS3 enables hardware acceleration on some platforms (notably WebKit based browsers such as Mobile Safari). When using this approach, elements are HTML DOM elements like anything else on an HTML page. Browser support is improving all the time and the features needed by AV content are supported by most desktop browsers (IE9 will support) and latest mobile browsers.

[5] `http://blog.scratch.mit.edu/2010/04/scratch-on-iphone.html` (visited July 5, 2011)
[6] UK, France, Germany, Italy and Spain
[7] `http://raphaeljs.com/` (visited July 5, 2011)
[8] `http://code.google.com/p/android/issues/detail?id=1376` (visited July 5, 2011)
[9] See `http://villekaravirta.com/?p=23` (visited July 5, 2011) for examples of sorting algorithms with CSS3.

In all the approaches, the interaction and control of the animation is done using JavaScript. This makes complex interactions possible. The JavaScript engines of all modern browsers are of high quality and the performance has improved significantly in the recent years.

Of the graphics approaches, canvas is the least well applicable for AV content. The main reason is that it is bitmap graphics which makes it less suited for scaling to different sizes. Furthermore, the lack of DOM elements and thus DOM events makes things unnecessarily difficult for developer. For SVG and CSS3 approaches it is more difficult to find differences. Both SVG animations and CSS3 transitions are supported equally well[10]. Both support CSS3 media queries to style the content according to, for example, device screen width/height. I see either choice to be good and the choice should probably be abstract away by a JavaScript library.

However, if building an AV system from scratch, I lean on HTML+CSS3 since data structures (the main parts in algorithm visualizations) are not actually graphics. Using HTML we can include some semantics with proper use of HTML elements such as a hierarchy of `div` elements for (binary) trees. If more complex graphics are needed than what CSS3 is naturally suited for, an overlay of SVG to add graphics could be used. Alternatively, the graphics can be simulated with advanced CSS3[11].

Cross-Device Development Platforms

It is worth mentioning that there are tools available to develop native applications that work across multiple platforms using web technologies. Tools such as Rhomobile[12], Phonegap[13], and Titanium[14] enable running and submitting to application stores applications built using web technologies. Furthermore, they provide access to native APIs like calendar and contact information through JavaScript. However, these do not remove the problem that more and more application stores are popping up and it is difficult to choose the right ones.

7.5 New Ways of Engagement on Mobile

The engagement taxonomy (Naps et al., 2003a) has provided a target for developers to add interaction to visualization systems. The engagement taxonomy specified five levels of engagement: viewing, responding, changing, constructing, and presenting. While introducing the levels is unnecessary for the probable audience of this paper, the following will, however, consider how these engagement levels could be provided in mobile contexts and what special requirements there are for them. Furthermore, the latest devices offer some interesting possibilities for innovation, and those are discussed as well.

- **Viewing:** In general, viewing on mobile is no different from viewing on desktop AVs. However, in mobile devices, it is important not to require scrollbars when viewing content, so the content shown needs to fit a single screen. Furthermore, the controls for the animation require space, and it will benefit to consider which of the controls are actually necessary.

- **Responding:** Responding is often implemented as pop-up questions about the content of the visualization. This is difficult with limited screen estate, as it would require fitting the question, answers, and the visualization needed to answer on one screen. One way to solve this is to enable answering the question by interacting with the visualization.

- **Changing:** Changing in the sense of providing input data can work quite similarly to traditional AVs.

- **Constructing:** The touch interfaces alone make interacting with the visualization different than using a mouse. For example, drag-and-drop is somewhat more difficult, and a lot of the UI operations need to be though again. For example, simulation assignments similar to those in TRAKLA2 (Malmi et al., 2004) need to be redesigned. *Multitouch gestures* (that is, using multiple fingers on the screen) could be used to interact with the visualization. As just one example, rotation in a balanced search tree could be done with a rotating motion using two fingers.

[10] According to `http://caniuse.com/` (visited July 5, 2011)

[11] See `http://css-tricks.com/examples/ShapesOfCSS/` and `http://lab.simurai.com/toy/monster/` for examples (visited July 5, 2011).

[12] `http://rhomobile.com/` (visited July 5, 2011)

[13] `http://www.phonegap.com/` (visited July 5, 2011)

[14] `http://www.appcelerator.com/products/titanium-mobile-application-development/` (visited July 5, 2011)

- **Presenting:** Presenting in the traditional sense is not really applicable with today's devices. However, the video and voice recording capabilities do offer some interesting possibilities for the future. Imagine a mobile application used to view an AV and, at the same time, recording students explanation and publishing the result online as a podcast or screencast.

Furthermore, new input devices like accelometers measuring the position and acceleration of the device could be used in creative ways to make the AVs more game-like. Imagine, for example, a labyrinth-like game of search from a binary search tree where student needs to guide the item to be inserted to correct position by tilting the phone. Or, providing random input for an algorithm by shaking the phone.

Constant internet connection of the devices could also open up new possibilities in the form of multiplayer algorithm visualizations that could provide collaboration and competition as a form of engagement.

7.6 Future Research Challenges

Based on the discussion above and due to the lack of space to go through these in this paper, we conclude the paper with open research challenges for the AV community to work on[15].

- Design for mobile requires to simplify and consider what is necessary for the current task. Which of the requirements for AV are requirements for mobile AV?
- How to evaluate visualizations on mobile devices? Naps et al. (2003b) provide guidelines for evaluating AV. Which are suitable for evaluation of mobile AVs?
- How to combine AVs into interactive learning materials and integrate them with hypertext on mobile?
- How can we reuse and adapt the existing AV content for mobile?
- How to author AVs suitable for different devices? Should there be AV authoring tools that work on mobile devices?

To summarize, I see mobile devices as an important future development direction for algorithm visualization research. I hope this paper will motivate others to pursue development in that direction as well.

[15] It is no surprise that many of these are still open research questions for traditional AVs as well.

Bibliography

Christopher M. Boroni, Frances W. Goosey, Michael T. Grinder, and Rockford J. Ross. A paradigm shift! the internet, the web, browsers, java and the future of computer science education. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education*, pages 145–152, New York, NY, USA, 1998. ACM.

Brian Fling. *Mobile Design and Development: Practical Concepts and Techniques for Creating Mobile Sites and Web Apps*. O'Reilly Media, 2009.

Eva Patricia Gil-Rodríguez and Pablo Rebaque-Rivas. Mobile learning and commuting: contextual interview and design of mobile scenarios. In *Proceedings of the 6th International Conference on HCI in Work and Learning, Life and Leisure: Workgroup Human-Computer Interaction and Usability Engineering*, pages 266–277, Berlin, Heidelberg, 2010. Springer-Verlag.

Ville Karavirta, Ari Korhonen, Lauri Malmi, and Kimmo Stålnacke. MatrixPro – A tool for on-the-fly demonstration of data structures and algorithms. In *Proceedings of the 3rd Program Visualization Workshop*, pages 26–33, The University of Warwick, UK, July 2004. University of Warwick, UK.

Jeanne Lam, Jane Yau, and Simon K. S. Cheung. A review of mobile learning in the mobile age. In *Proceedings of the 3rd International Conference on Hybrid Learning*, pages 306–315, Berlin, Heidelberg, 2010. Springer-Verlag.

Lauri Malmi and Ari Korhonen. Automatic feedback and resubmissions as learning aid. In *Proceedings of 4th IEEE International Conference on Advanced Learning Technologies*, pages 186–190, Joensuu, Finland, 2004. IEEE.

Lauri Malmi, Ville Karavirta, Ari Korhonen, Jussi Nikander, Otto Seppälä, and Panu Silvasti. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education*, 3(2):267–288, 2004.

Thomas L. Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, and J. Ángel Velázquez-Iturbide. Exploring the role of visualization and engagement in computer science education. *SIGCSE Bulletin*, 35(2):131–152, June 2003a.

Thomas L. Naps, Guido Rößling, Jay Anderson, Stephen Cooper, Wanda Dann, Rudolf Fleischer, Boris Koldehofe, Ari Korhonen, Marja Kuittinen, Charles Leska, Lauri Malmi, Myles McNally, Jarmo Rantakokko, and Rockford J. Ross. Evaluating the educational impact of visualization. *SIGCSE Bulletin*, 35(4):124–136, December 2003b.

Daniel Su Kuen Seong. Usability guidelines for designing mobile learning portals. In *Proceedings of the 3rd International Conference on Mobile Technology, Applications & Systems*, New York, NY, USA, 2006. ACM.

Clifford A. Shaffer, Matthew L. Cooper, Alexander Joel D. Alon, Monika Akbar, Michael Stewart, Sean Ponce, and Stephen H. Edwards. Algorithm visualization: The state of the field. *Transactions on Computing Education*, 10:9:1–9:22, August 2010.

Sushil K. Sharma and Fred L. Kitchens. Web services architecture for m-learning. *Electronic Journal on e-Learning*, 2(1):203–216, February 2004.

8 Initial Set of Services for Algorithm Visualization

Ville Karavirta and Petri Ihantola
Aalto University, School of Science
Department of Computer Science and Engineering

ville.karavirta@aalto.fi, petri.ihantola@aalto.fi

8.1 Introduction

When considering algorithm visualization (AV) systems, there are parts of the software that each system implement separately, but with ultimately the same functionality. Examples of this include graph drawing (i.e. calculating positions of nodes for a graph to draw it esthetically) and hypertext integration. Pop-up questions are another common feature and Rößling and Häussage (2004) have actually implemented a tool-independent Java library to answer this need. In addition, Rößling (2009) has introduced design patterns for implementing some common features.

The benefits of service oriented architectures include the reuse of components, ease of exposing and integrating legacy systems as services, loose coupling of components, and more rapid development of new systems by service composition. We feel that since most of the AV systems work online nowadays, it would make sense to provide the common parts of AV software as web services.

In this work in progress paper, our aim is to introduce the idea of using services to compose/support AV systems. We will do this by describing an initial set of services we have implemented (see Sections 8.2 and 8.3). We also describe some third party services that can be used for AV (Section 8.4). Furthermore, we will discuss possible other parts of existing AV systems that could very well be exposed as services for others to benefit (Section 8.5).

8.2 Service for Content Embedding

The Problem:

There has been a lot of research into integrating AV into hypertext (Rößling et al., 2006; Karavirta, 2009). Most of the systems have solved this by using Java applets (Shaffer et al., 2010). Another approach has been taken by using HTML and JavaScript (Karavirta, 2009). Neither of these approaches, however, solve the problem of incorporating third party AVs found online into teacher's course material.

The Service Solution:

OEmbed[1] is a specification defining how to embed third party content into ones own HTML pages. It is widely used with popular services such as YouTube and Flickr, which both offer an OEmbed service. The general idea is to make an HTTP request to an OEmbed service giving the URL of the content to be embedded as a URL parameter. The response can then be, for example, JSON (JavaScript Object Notation) or XML and it describes the content in the requested URL.

The specification defines a set of parameters for the request, mainly the URL of the content, as well as optional parameters maxwidth, maxheight, and format for controlling the width and height of the content and the response format, respectively. For the response, the specification has two required parameters: type (the type of content, such as photo, video, or rich) and version (of the OEmbed specification used). Optional parameters include, for example, title, author and provider information, thumbnail url, and caching age.

We have implemented an OEmbed service for the algorithm visualizations in the TRAKLA2 system (Malmi et al., 2004). It allows teachers to embed TRAKLA2 exercise applets[2] from our servers to any HTML learning material.

[1] http://www.oembed.com/

[2] Those that have a resource URL at http://www.cse.hut.fi/en/research/SVG/TRAKLA2/exercises.shtml.

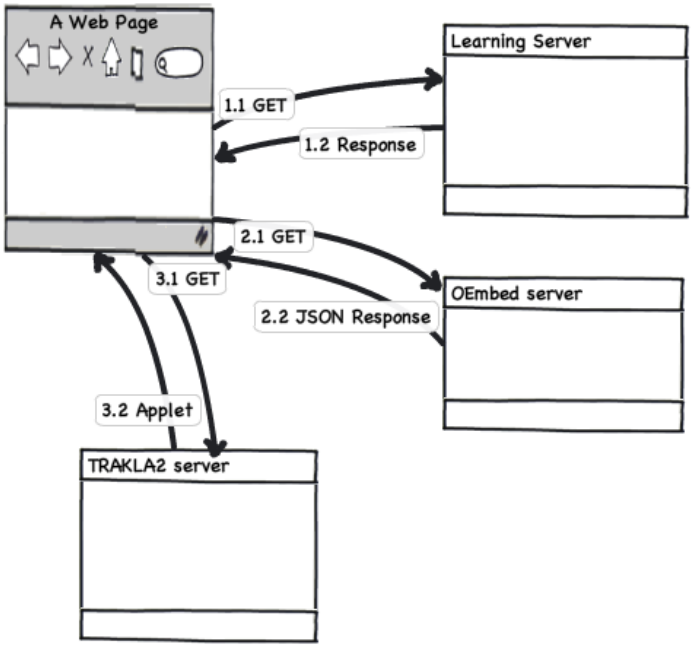


Figure 8.1: The process of calling OEmbed service when embedding content to learning material.

Example Usage:

The way the service works is explained by Figure 8.1 for the case of calling the OEmbed server from the student's browser. First, student requests and gets the learning material from the server (step 1.1 and 1.2). This HTML includes JavaScript to call the OEmbed server (step 2.1), in our case to the following URL:

`http://example.com/oembed/?url=http://www.cse.hut.fi/en/research/SVG/TRAKLA2/exercises/BST_TraverseIn-9.html`

which responds with the following JSON (step 2.2).

```
  {"version": "1.0",
   "type": "rich",
   "html": "<applet␣code=\"applications.exerciseapplet.ExerciseApplet.class\"
␣␣␣archive=\"http://www.cse.hut.fi/en/research/SVG/TRAKLA2/exercises/matrix.jar\"
␣␣␣width=\"650\"␣height=\"600\">
␣␣␣␣<param␣name=\"ex\"␣value=\"content.exercises.BST_TraverseIn\"␣/>
␣␣␣␣<param␣name=\"reset\"␣value=\"1\"␣/>
␣␣␣␣</applet>"}
```

From this response, the HTML to display the exercise applet can be extracted and inserted into the page. When that is done, the browser will fetch the actual applet from TRAKLA2 server (step 3.1 and 3.2).

The call to the OEmbed server can be done either by a server-side software or by JavaScript running in the browser. To get around the same-origin policy of JavaScript sandbox in browsers, the service endpoint supports an optional parameter `callback`. Another parameter that the service accepts is `format` which can be used to specify XML response by setting `format=xml`.

Discussion:

The service we have introduced works for the TRAKLA2 content. In the larger scale of algorithm visualizations, this is a rather limited collection. In the future, an OEmbed service could be integrated with the AlgoViz Catalog that has a collection of numerous AVs (Shaffer et al., 2010). There, it could provide a general way to have a repository of AVs that can be easily and consistently embedded to hypertext learning materials. In this context, it would also make sense to include more metadata supported by the OEmbed specification, such as author and provider information.

Another future extension would be more TRAKLA2 specific. The embedded applet could also store the student submissions on the server (as the actual TRAKLA2 exercise environment does). The way this could work is for the teacher to register for an application key that is provided in the request together with student ID to identify the course and student.

8.3 Service for Graph Layout

The Problem:

Calculating the positions of nodes in a graph is a difficult problem that needs to be solved by every AV system that visualizes graphs. The goal is to be able to draw the graph as aesthetically pleasingly as possible, for example, by minimizing the number of overlapping nodes and edges. As also pointed out by Hamer (2004), there is no reason for every AV system to implement graph drawing algorithms separately. In his standalone Lightweight Java Visualization tool, Hamer used Graphviz[3] to externalize graph layout and drawing.

The Service Solution:

Our service for graph drawing consists of just one URL that can be called with the graph edge list as a parameter. The response from the service can be XML or JSON and it includes `x` and `y` coordinates for all the nodes in the graph. For the JSON format, an additional `callback` parameter can be passed to enable cross-domain callback function. In addition, to support layouts of different size restrictions, optional `w` and `h` parameters can be passed to specify the width and height of the layout area, respectively.

The described service is implemented using Django web framework[4] and NetworkX (Hagberg et al., 2008) software package. NetworkX is a Python software for manipulating and working with network structures and it offers a wide range of algorithms for analyzing and drawing graphs.

Example Usage:

To give an example of using the service, a request to:
`http://example.com/draw?graph=a-b,b-c,c-d,d-a`
will return JSON response like the following.

```
{"status": "OK",
 "coordinates": {
    "a": {"y": 43, "x": 100},
    "b": {"y": 100, "x": 55},
    "c": {"y": 56, "x": 0},
    "d": {"y": 0, "x": 43
 }
}
```

In the response, there are x and y coordinates for all the nodes in the graph. The assumption is that edges are straight lines connecting the nodes.

Discussion:

To be more useful, an important feature would be to allow requesting coordinates for a series of graph states to be used in algorithms where the graph structure is changed as the algorithm proceeds. This would decrease the number of requests made, as well as improve the quality of the results by allowing the layout algorithm to take advantage of coordinates calculated for the previous states.

The advantage of the service compared to using the Google Chart API (see Section 8.4.1) is that layout is easy to isolate from a system. Furthermore, since the rendering is still done by the AV system, adding interaction is easier than it would be if the service rendered images.

Graph layout can be implemented by including a graph drawing library into the system. Even the same NetworkX software could be included in applets by using Jython (Python for Java platform, `http://jython.org/`). The downside of this is that transferring all that data for something easily achievable and isolatable as a service is unnecessary. This is even more important once AV systems working on mobile devices are developed.

8.4 Existing Services for AV

Some services exist that could be used in algorithm visualization systems. Examples of such third party services are introduced below.

[3] `http://www.graphviz.org`
[4] `http://www.djangoproject.com/`

8.4.1 Graph Images

Images of data structures are often needed in learning material related to data structures and algorithms. Google Chart API[5] can render latex, graphs, and many other diagrams. However, from the AV point of view, experimental Graphviz support is the most interesting feature. You can use Graphviz's DOT language as an intermediate presentation and pass this to the service. Service provides various layout engines to choose from, and produces images based on the input. With the service, graph description is part of URL, as in `https://chart.googleapis.com/chart?cht=gv&chl=graph{a--b--c;b--d}`. Callbacks and controlling the size of a requested image are both also supported.

In addition to rendering images, Chart API can also return image maps in JSON. This is designed to allow interaction with parts of an image. In our previous example, for example, you could get the location of node b and write your own JavaScript of what should happen when the cursor is on top of that node. Another use case for image maps is to misuse them as layout engines like our graph layout service and let another part of the software do the drawing.

8.4.2 User Modeling

User modeling is an important part in adaptive hypermedia systems. Adaptivity has been an underused feature in AV systems, most probably due to the difficulty of making it good. The most notable example of adaptivity in AV is WadeIn II (Brusilovsky and Loboda, 2006).

A user modeling server/service named CUMULATE (Brusilovsky et al., 2005) and its follower CUMULATE 2 (Yudelson et al., 2007) have been presented. The server provides services for reporting user activity and changes in user knowledge as well as requesting user progress and knowledge of concepts. The server requested data is delivered either as XML or as serialized Java objects. In addition, there is work in progress to serialize the CUMULATE data model as RDF (Resource Description Format). For more details, see the CUMULATE wikipage[6]. The downside of CUMULATE is that, as far as we know, the server is not available for local installation.

8.5 What Other Services are Needed

While we see the two services we have already implemented solving some of the most difficult (graph layout) and typical (embedding of AV) problems, there are other potential services for algorithm visualization as well. The following subsections will introduce them.

8.5.1 Initial Data Generation

Generating initial data for various algorithms to work on is a tedious task for humans. Implementing automated generators for good initial data takes a lot of effort as well. The generation code needs to be implemented for each algorithm to make sure all the special cases (or the ones wanted by the teacher) will be shown by the data. For example, in the case of AVL tree, all kinds of rotations should be encountered by the student.

Systems like JHAVE (Naps, 2005) and TRAKLA2 have a lot of code that generate different, pedagogically sensible initial data for various algorithms. These initial data generators could be exposed as services for other AV systems to use. This would aid in generating new visualizations for the algorithms in other use cases.

8.5.2 Content Transformations

Many AV systems have textual formats for describing the animation content (Karavirta et al., 2010). Previous work has aimed at enabling transformations of these formats to enable reuse of AVs in different systems (Karavirta, 2007). These transformations could very well be exposed as services to enable content reuse. Again, such a service transforming AV content could very well be integrated with an AV repository like the AlgoViz Catalog.

[5] `http://code.google.com/apis/chart/`

[6] `http://adapt2.sis.pitt.edu/wiki/CUMULATE`

8.5.3 Content Generators

ANIMAL system includes a large collection of configurable content generators for various algorithms (Rößling and Ackermann, 2006). Exposing these as services that can be called through HTTP requests would enable using them in more versatile environments than using the Java application. Furthermore, together with the content transformation service, it could provide a composite service enabling use of content generators and getting the resulting AV in multiple formats.

8.6 Discussion and Conclusions

Services enable rapid development and easier deployment of new AV and educational web software through service composition. In this paper, we have presented two such services for calculating graph layouts and for embedding AVs into hypertext.

Trusting a service hosted by others can always be a difficult decision to make. Teachers need to carefully consider whether the uptime of such hosted services is high enough. In addition, student information such as user modeling cannot be transferred to any service. Thus, we strongly believe all AV supporting services should be open sourced to enable local installs. This also enables shared development of common codebase. Code for the implemented services presented in this paper will be open sourced, most probably on GitHub[7].

In addition to our own initial set of services, we have explored some existing, third party services also valuable for AV developers. We have also presented ideas for other possible services needed in the future. We hope that someone tackles the challenge and extract those features from their favorite standalone programs and turn those into services.

[7] http://github.com/

Bibliography

Peter Brusilovsky and Tomasz D. Loboda. WADEIn II: a case for adaptive explanatory visualization. In *Proceedings of the 11th annual SIGCSE conference on Innovation and Technology in Computer Science Education, ITICSE'06*, pages 48–52, New York, NY, USA, 2006. ACM. ISBN 1-59593-055-8. doi: http://doi.acm.org/10.1145/1140124.1140140.

Peter Brusilovsky, Sergey Sosnovsky, and Olena Shcherbinina. User Modeling in a Distributed E-Learning Architecture. pages 387–391. 2005. doi: 10.1007/11527886_50. URL `http://dx.doi.org/10.1007/11527886_50`.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008. URL `http://conference.scipy.org/proceedings/SciPy2008/paper_2/`.

John Hamer. A lightweight visualizer for java. In *Proceedings of Third Program Visualization Workshop*, pages 54–61, 2004.

Ville Karavirta. Integrating algorithm visualization systems. In *Proceedings of the Fourth Program Visualization Workshop (PVW 2006)*, volume 178 of *Electronic Notes in Theoretical Computer Science*, pages 79–87, Amsterdam, The Netherlands, The Netherlands, 2007. Elsevier Science Publishers B. V.

Ville Karavirta. Seamless merging of hypertext and algorithm animation. *ACM Transactions on Computing Education (TOCE)*, 9(2):1–18, 2009. URL `http://doi.acm.org/10.1145/1538234.1538237`.

Ville Karavirta, Ari Korhonen, Lauri Malmi, and Thomas Naps. A comprehensive taxonomy of algorithm animation languages. *Journal of Visual Languages & Computing*, 21(1):1–22, 2010. ISSN 1045-926X. doi: DOI:10.1016/j.jvlc.2009.09.001. URL `http://dx.doi.org/10.1016/j.jvlc.2009.09.001`.

Lauri Malmi, Ville Karavirta, Ari Korhonen, Jussi Nikander, Otto Seppälä, and Panu Silvasti. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education*, 3(2):267–288, 2004.

Thomas L. Naps. JHAVÉ: Supporting Algorithm Visualization. *Computer Graphics and Applications, IEEE*, 25(5):49–55, 2005.

Guido Rößling. A first set of design patterns for algorithm animation. In *Proceedings of the Fifth Program Visualization Workshop, PVW'08*, volume 224 of *Electronic Notes in Theoretical Computer Science*, pages 67–76, Amsterdam, The Netherlands, 2009. Elsevier Science Publishers B. V. doi: DOI:10.1016/j.entcs.2008.12.050. URL `http://www.sciencedirect.com/science/article/B75H1-4VBD6P0-8/2/1b9b0b90c2285d941126fdea31648829`.

Guido Rößling and Tobias Ackermann. A framework for generating AV content on-the-fly. In *Proceedings of the Fourth Program Visualization Workshop*, 2006.

Guido Rößling and Gina Häussage. Towards tool-independent interaction support. In *Proceedings of the Third Program Visualization Workshop*, pages 110–117, The University of Warwick, UK, July 2004.

Guido Rößling, Thomas Naps, Mark S. Hall, Ville Karavirta, Andreas Kerren, Charles Leska, Andrés Moreno, Rainer Oechsle, Susan H. Rodger, Jaime Urquiza-Fuentes, and J. Ángel Velázquez-Iturbide. Merging interactive visualizations with hypertextbooks and course management. *SIGCSE Bulletin*, 38 (4):166–181, 2006. ISSN 0097-8418. URL `http://doi.acm.org/10.1145/1189136.1189184`.

Clifford A. Shaffer, Matthew L. Cooper, Alexander Joel D. Alon, Monika Akbar, Michael Stewart, Sean Ponce, and Stephen H. Edwards. Algorithm visualization: The state of the field. *Trans. Comput. Educ.*, 10:9:1–9:22, August 2010. ISSN 1946-6226. doi: http://doi.acm.org/10.1145/1821996.1821997. URL `http://doi.acm.org/10.1145/1821996.1821997`.

Michael Yudelson, Peter Brusilovsky, and Vladimir Zadorozhny. A user modeling server for contemporary adaptive hypermedia: An evaluation of the push approach to evidence propagation. In *Proceedings of the 11th international conference on User Modeling*, UM '07, pages 27–36, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-73077-4. doi: http://dx.doi.org/10.1007/978-3-540-73078-1_6. URL http://dx.doi.org/10.1007/978-3-540-73078-1_6.

9 Context-Sensitive Guidance in the UUhistle Program Visualization System

Juha Sorva, Teemu Sirkiä
Aalto University, School of Science, Espoo, Finland

{juha.sorva, teemu.sirkia}@aalto.fi

9.1 Introduction: UUhistle and Visual Program Simulation

UUhistle (pronounced "whistle") is a program visualization (PV) system for introductory programming education. It is meant for visualizing the execution of small Python programs. UUhistle's pedagogical goals include promoting a dynamic view of program behavior, easing the learning of program tracing, and addressing misconceptions of programming concepts. UUhistle can be used for a number of activities that include the following.

- *Debugging with animations.* Learners can view animations of the execution of programs that they wrote themselves. In this usage, UUhistle serves as a graphical, very detailed sort of debugger.
- *Exploring examples.* Learners can view animations of teacher-given programs. Teachers may configure UUhistle to adjust how particular example programs are shown.
- *Interactive program animation.* UUhistle can animate the execution of programs created on the fly so that the execution of each instruction is shown as soon as the instruction is typed in (Figure 9.1).
- *Visual program simulation.* Learners can engage in visual program simulation exercises in which they manually carry out the execution of an existing program (Figure 9.2). UUhistle gives automatic feedback on visual program simulation exercises and can automatically grade students' solutions.
- *Presentations.* Teachers and learners can use UUhistle's animations as an aid when discussing the execution-time behavior of programs.
- *Quizzes.* Stop-and-think questions embedded into example programs for learners to answer.

UUhistle has been designed to support active learning and specifically to encourage interaction between learner and visualization. One of UUhistle's more noteworthy features is its support for visual program simulation (VPS). In a VPS exercise, the learner takes the driving seat. Instead of watching an animation of what the computer does, the learner has to read given code and manipulate UUhistle's visualization to simulate the execution of instructions. Through the visualization, the learner creates variables and objects in memory, evaluates expressions, assigns values, manipulates the call stack, passes parameters, and so forth.

For instance, to evaluate the Python statement `c = a + b` in a VPS exercise, the learner is expected to drag and drop the first operand, the plus operator and the second operand to an expression evaluation area, then click on the plus to perform the operation and cause the result to appear in the expression evaluation area. The next step is to create the variable `c` in a context menu associated with the topmost frame of the call stack (assuming the variable didn't exist yet). Finally, the learner should drag the contents of the expression evaluation area (that is, the sum) into the target variable.

Any example program that UUhistle can visualize can be turned into a VPS exercise. Figure 9.2 shows a VPS exercise in UUhistle.

UUhistle's overall functionality, its theoretical background and a review of related systems we have presented elsewhere (Sorva and Sirkiä, 2010, and references therein). In this work-in-progress report we describe some new features of UUhistle. In particular, we focus on materials that UUhistle can present to the learner in a context-sensitive manner in order to address programming misconceptions and for other purposes.

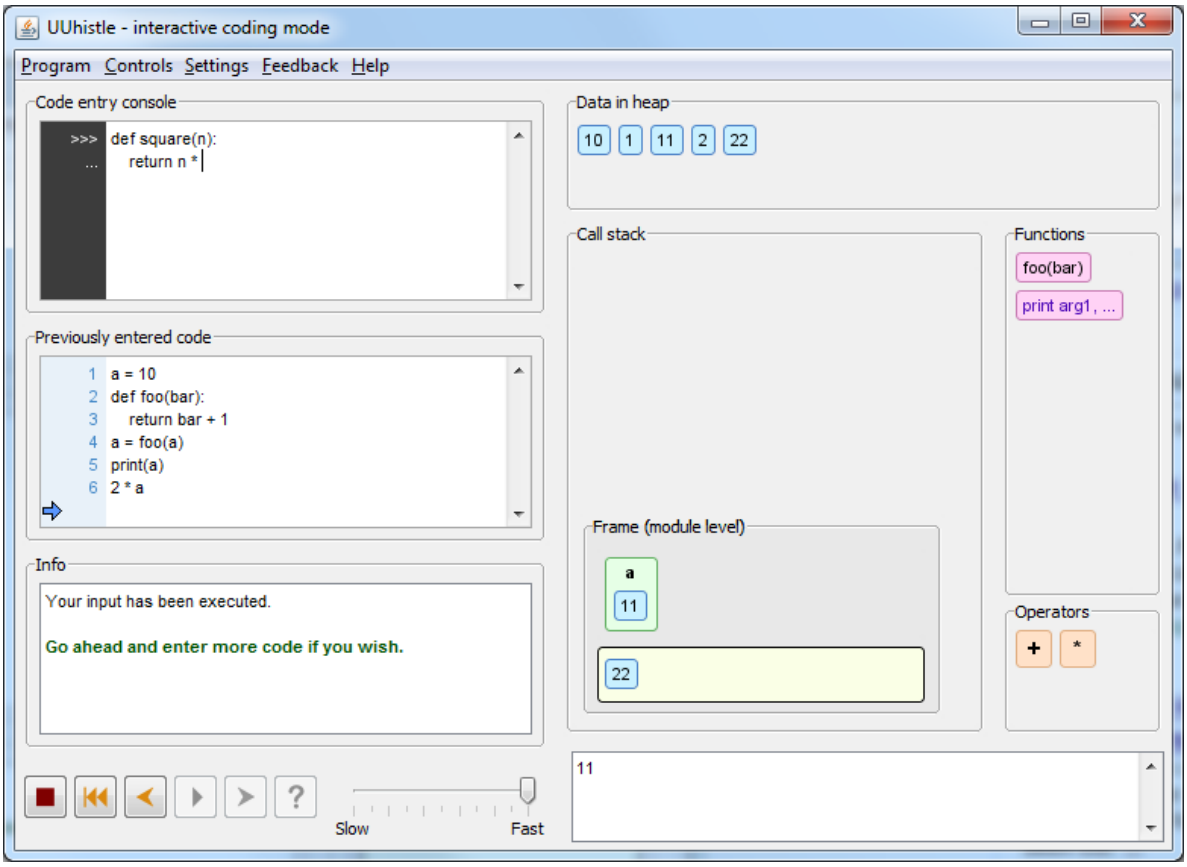


Figure 9.1: UUhistle visualizes contents of memory as abstract graphics. Here, UUhistle is in interactive coding mode. The user is typing in a function in the top left corner. UUhistle visually executes each new input immediately.

9.2 Addressing Misconceptions

One of the goals of various PV tools for CS1 – ours, too – is to address novice misconceptions. Ben-Ari (2001) leavens our enthusiasm with a cognitive constructivist point of view:

> *For the use of [an educational software visualization system] to be effective, the teacher must (a) have a clear idea of the existing mental model of the student, (b) specify in advance the characteristics of the mental models that the instruction is intended to produce, and (c) explain exactly how the visualization will be instrumental in effecting the transition. This is an immensely difficult undertaking. [...] the existing mental models of the individual students are different and it takes quite a lot of effort to elicit even an approximation of a cognitive structure.*

Ben-Ari's statement underlines the usefulness of knowing about the mental models (and misconceptions) of individual students. Eliciting these from each student is indeed immensely difficult. A couple of things make life easier for the visualization designer, however. First, nature constrains our knowledge construction (Phillips, 2000). It is reasonable to expect that there are similarities between learners' mental models, and that certain misconceptions are more common than others. Second, creating a state of cognitive conflict does not necessarily and always require a clear idea of the learner's mental model. In some cases, it is surely possible to make the learner deal with a conceptual model (visualization) so that the learner him- or herself is in a position to discover a discrepancy between the conceptual model and his or her own prior mental model, whatever it is like.

That said, Ben-Ari's point is an important one. A visualization that directly and explicitly addresses a learner's misconception likely has a better chance of success than one that doesn't. But how can a fully

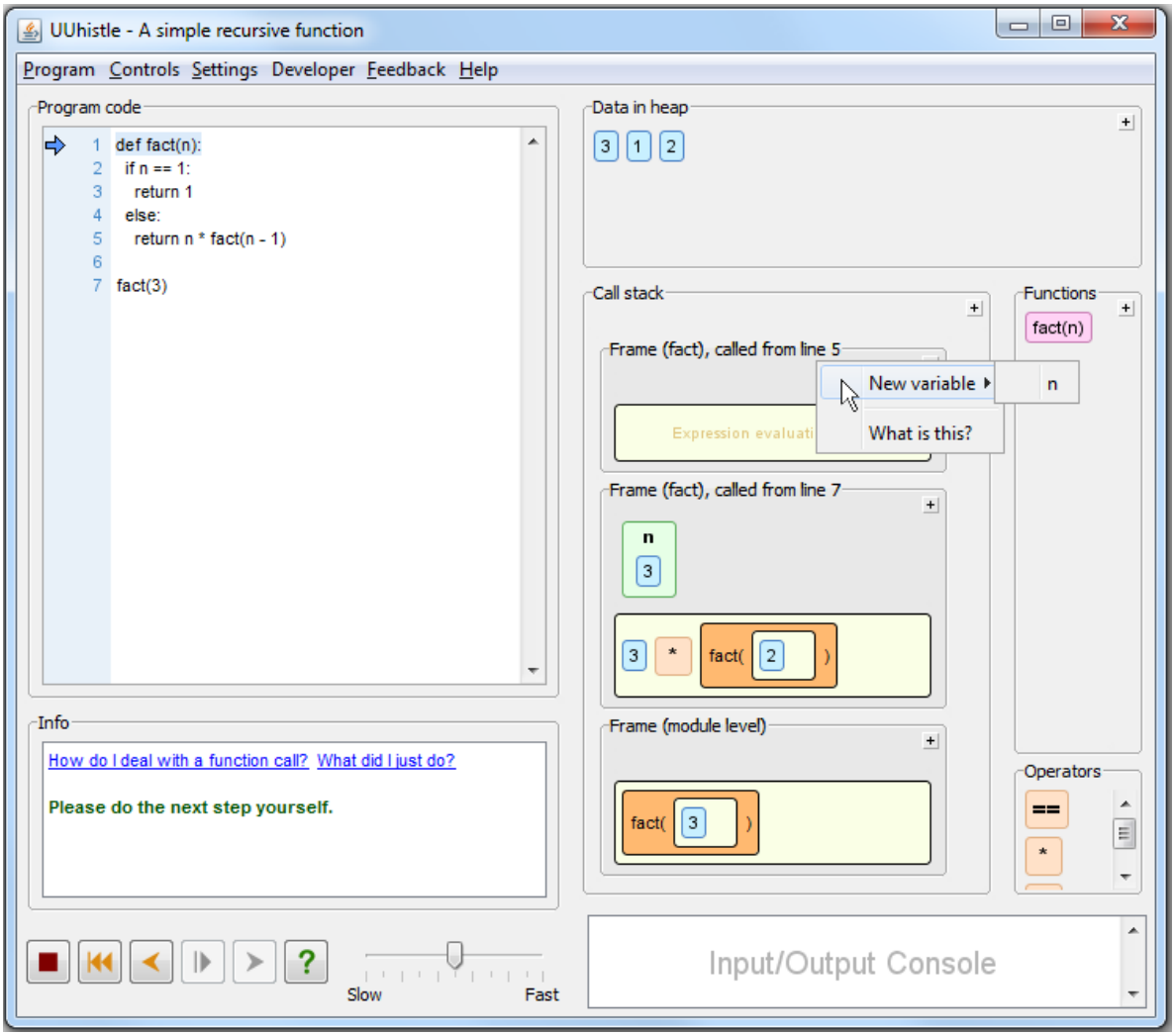


Figure 9.2: A VPS exercise in UUhistle. The user is some way into simulating the execution of a small recursive program, and has just clicked on the topmost frame in order to create a new variable there through the context menu. Next, the user is expected to drag and drop the parameter value 2 from the lower frame into the new variable. Control will automatically move to line 2; the user then continues simulating the program with a combination of mouse clicks and drag-and-drops.

automatic visualization system possibly address individual students' misconceptions? A visual program simulation system may fare better than most...

9.2.1 VPS and misconceptions

In a VPS exercise, the learner needs to pick the correct execution steps from among various alternatives. There is plenty of room for mistakes, as there should be, lest the VPS activity degenerate into mere *controlled viewing* (Myller et al., 2009) with an unwieldy user interface.

Learners may misunderstand the semantics of many programming constructs and pick the wrong kind of execution step. They may use the wrong values or the wrong operators. They may not allocate memory for frames or objects. They may do steps in the wrong order.

Some mistakes are rooted in misconceptions. For instance, the learner may fail to create another frame for a recursive call and instead reuse the current frame's variables. This mistake may be symptomatic of a "looping model" of recursion (see e.g. Götschi et al., 2003).

We have devised a five-level scheme that serves as one way of looking at how a VPS system may address a misconception that concerns program behavior. The levels are:

Level 0 **By doing nothing.** *The system does not address the misconception. Nothing in the visualization conflicts with the misconception.*

Level 1 **By showing what's right.** *The system shows that what actually happens when the program is run does not match the learner's misconception. If the learner pays enough attention, they may experience cognitive conflict between their understanding and the visualization. (A program animation that doesn't activate the learner and has no clue of what the learner thinks can also address misconceptions at this level.)*

Level 2 **By having the learner do what's right.** *The VPS system requires the learner to simulate aspects of program execution that pertain to the misconception. The system prevents making the kind of mistake the learner would likely make if they consistently followed through with their misconception. The learner has to perform simulation steps that go against their misconception. The system does not encourage the learner to contrast the accepted (correct) simulation step with any alternatives.*

Level 3 **By allowing what's wrong.** *The VPS system requires the learner to simulate aspects of program execution that pertain to the misconception. Further, the learner can follow through with their misconception by making an incorrect simulation step that corresponds to it. To solve the VPS exercise, the learner has to pick the correct execution step rather than one that matches their misconception. If the learner fails to do so, the system notifies the learner that something is wrong. Ideally, this alerts the learner to the shortcoming in their thinking even though the system cannot guess what the specific problem is.*

Level 4 **By discussing what's wrong.** *As above, except that the system gives the learner context-sensitive feedback that is tailored to the kind of misconception that the learner's mistake probably indicates. The system helps the learner reflect on the particular misconception.*

With this scheme, we aim to distinguish between different ways of addressing misconceptions, not to suggest that every misconception needs to be addressed at Level 4. Showing carefully selected program examples will sometimes suffice to address a misconception. Misconceptions can also be addressed during program-writing tasks, for instance. Not every misconception even *can* be addressed at the higher levels – the learner's understanding may be so vague or inconsistent that they can not follow through with it in a VPS exercise. Nonetheless, it seems a reasonable hypothesis that for many misconceptions, each higher level in this scheme is likelier to result in fruitful cognitive conflict than the lower ones.

9.2.2 UUhistle and misconceptions

An example of a misconception that UUhistle addresses at Level 0 – not at all – is the notion that the computer (always) keeps program output in memory as part of program state (Bayman and Mayer, 1983). This particular misconception in fact is one that UUhistle and other PV systems may encourage as they show a visualization of program state alongside output. Also at Level 0 are the misconceptions in which the learner thinks it is possible for the programmer to do something that isn't actually supported by the language. The learner may mistakenly think, for instance, that it is possible to write a method that replaces the acting object itself with another (Ragonis and Ben-Ari, 2005). UUhistle cannot demonstrate that such a command does not exist. Sometimes an increased understanding of existing commands may still indirectly show that the nonexistent commands are unnecessary for the learner's intended purpose.

Most of the known misconceptions that UUhistle's VPS exercises address at Level 1 have to do with control flow, which is largely automated in UUhistle on default settings (for convenience). Consider the misconception according to which whenever the conditional part of an `if` statement executed, anything that comes after the statement is not (du Boulay, 1986). This is shown to be incorrect by UUhistle's visualization, which the learner may observe from the way control moves from line to line. After the learner is done with executing the conditional part, UUhistle chooses the next line automatically. The learner is not in any way involved in the act of changing lines.

For an example of a misconception that UUhistle's VPS addresses at Level 2, consider the notion that a variable can hold multiple values at the same time (e.g. du Boulay, 1986) . This is in direct conflict with what happens in UUhistle when the user drags a new value to a variable. No matter how much the learner might expect to produce a variable with two values, it won't happen, nor will they find another GUI operation that makes it happen. The learner is 'forced' into performing a simulation step whose outcome contradicts their misconception. Another class of misconceptions incorrectly constrains the capabilities of

the programmer. For instance, the learner may believe that the attributes of simple objects must always be accessed through a composite object (Ragonis and Ben-Ari, 2005). Arguably, the main way – and often sufficient way – to address such misconceptions is to select examples so that they show variation in the ways in which the task can be accomplished. What VPS can add is the requirement for the learner to think their way through the execution of the counterexamples. A similar sort of misconception is the excessively narrow definition of a concept, e.g. an object is seen as just a wrapper for a single variable (Holland et al., 1997); here, too, VPS can require the learner to work through an example that shows that objects can be more than that.

At Level 3, UUhistle allows a misstep that matches the misconception but signals that there is a problem. Automatic feedback points out that a mistake was made, and may give some generic advice, but does not directly address the specific misconception. For instance, the common notion that assignment works in the opposite direction is reflected in the learner dragging a value in the wrong direction. UUhistle allows the learner to carry out this incorrect step and informs them that they need to think again. UUhistle does not at present address the misconception that (possibly) led to the mistake by specifically explaining the semantics of assignment statements. Also at Level 3 is UUhistle's response to the misconception that values are updated by the computer according to their logical context. The learner can try to reflect this understanding in how they simulate a program – by updating variables at wrong times – but will get an error message.

UUhistle's most recent version addresses a handful of misconceptions at Level 4. UUhistle attempts to identify these specific misconceptions and gives tailored feedback when the learner appears to have one of them. One of these misconceptions is that assigning an object (reference) creates a copy of the assigned object. Figure 9.3 shows UUhistle's feedback for this particular case. The looping model of recursion mentioned above is another misconception for which UUhistle gives tailored feedback.

Adding more Level 4 guidance for the user is one of the main priorities in UUhistle's development at the present time. We've compiled a (currently unpublished) list of over 150 misconceptions, partial understandings and similar difficulties that novice programmers have according to the literature. UUhistle currently addresses the majority of these problems at Level 2 or 3. In our tentative estimate, at least a few dozen of the misconceptions look 'promising' in the sense that we have a reasonable guess of exactly what the user is likely to do in VPS if they consistently follow the misconception. An example is the assigning-in-the-wrong-directions misconception mentioned above. These misconceptions could perhaps be fairly reliably detected by UUhistle based on the learner's VPS actions, and could be addressed automatically with context-sensitive, misconception-aware feedback.

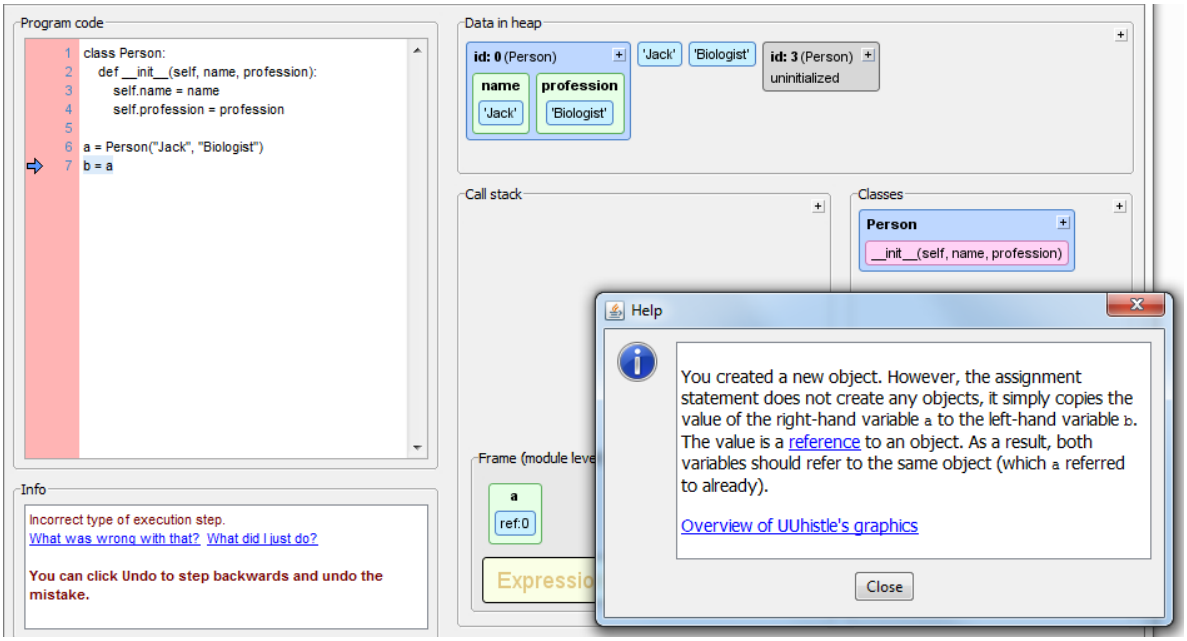


Figure 9.3: The user has created a new object when they should have merely formed another reference to an existing object. UUhistle's feedback – which the user accesses by asking "What was wrong with that?" – seeks to address a suspected misconception concerning object assignment.

9.3 Other Forms of Context-Sensitive Guidance

Apart from addressing known misconceptions, we are in the process of adding other context-sensitive materials to UUhistle. Under specific conditions, UUhistle's Info box invites the learner to study texts that accompany the visualization.

Figure 9.4 shows an example of VPS feedback which is tailored to a particular sort of mistake but which hasn't been designed to address a known misconception.

Figure 9.5 is another example of context-dependent guidance. UUhistle's VPS exercises are not invulnerable to guessing and naïve trial-and-error strategies, a situation which is not helped by the instant feedback that the system gives. Naïve approaches are laborsome because of the fairly large number of options from which to pick simulation steps, but the persistent student may still rely on them. When the user has trouble making progress – perhaps due to a naïve strategy – UUhistle attempts to encourage a more reflective approach.

Figure 9.6 is an example of how UUhistle seeks to alert the user to an interesting aspect of program execution: multiple references point to an object whose state has just changed. Here are a few other examples. When a function call is in progress, a link gives access to an overview of the stages of a function call. When two variables with the same name appear in different stack frames, UUhistle provides a link to a text that explains how each frame has its own variables. When a function is called for the second time, creating (again) a new frame, a link leads to a text that underlines the difference between a function definition and a function call, and discusses why it isn't practical to keep in memory a single reusable frame corresponding to each function definition (as students sometimes suggest). UUhistle highlights such points of interest both in VPS exercises and in regular program animations.

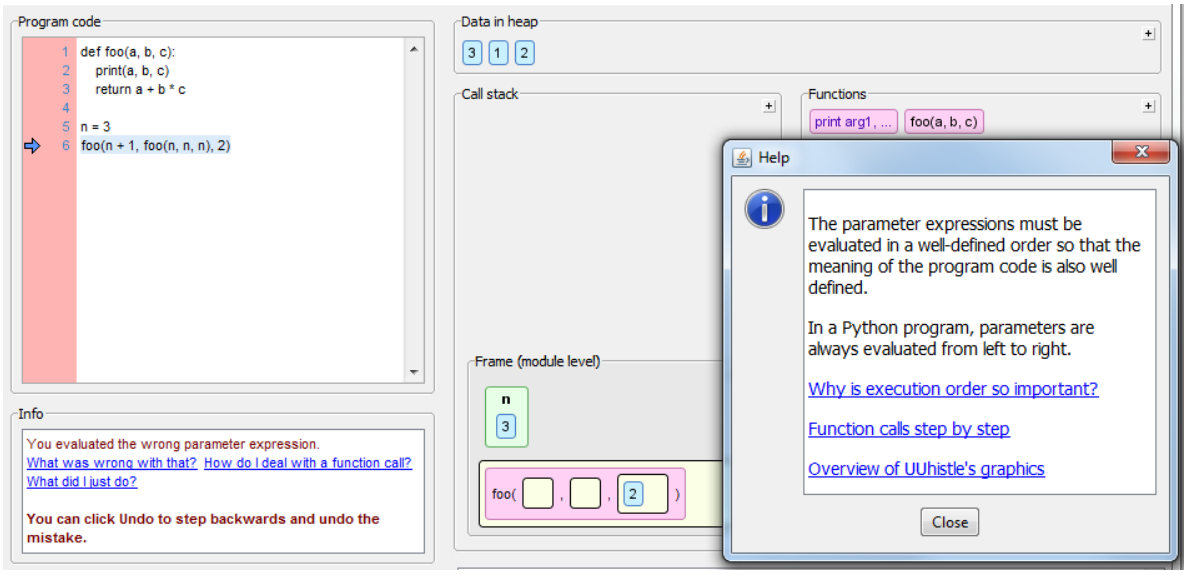


Figure 9.4: The user is evaluating parameters in non-standard order. Possibly they have thought that the order doesn't matter and they might as well start with the simplest parameter expression.

9.4 Conclusions

In this paper, we have presented some current directions in the development of the UUhistle program visualization system. Most importantly, we have described how we have developed UUhistle's visual program simulation exercises so that the system attempts to recognize specific misconceptions that novice programmers have and to provide feedback tailored to address those misconceptions. The effectiveness of this work in progress remains to be evaluated.

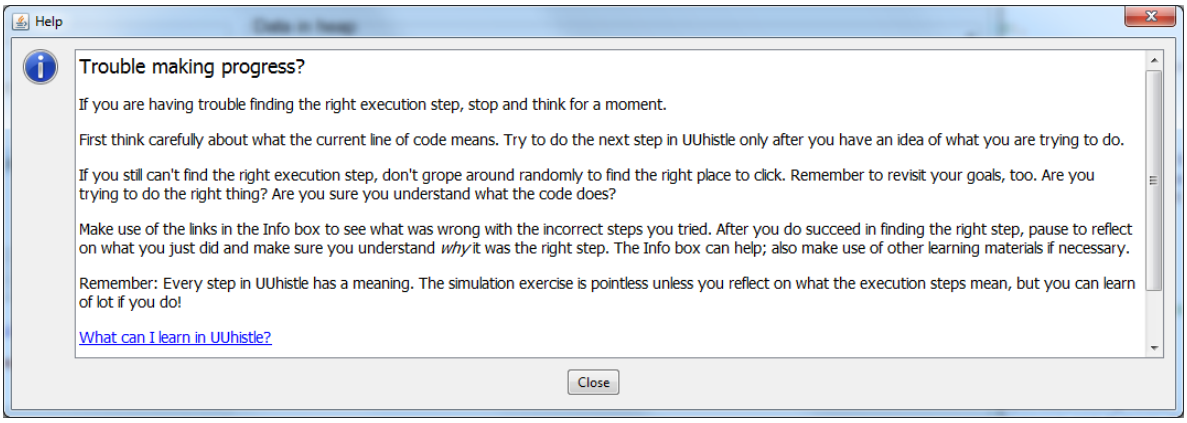


Figure 9.5: An "I have trouble making progress" link that brings up this dialog appears in the Info box if the user makes two successive failed attempts to find the next step.

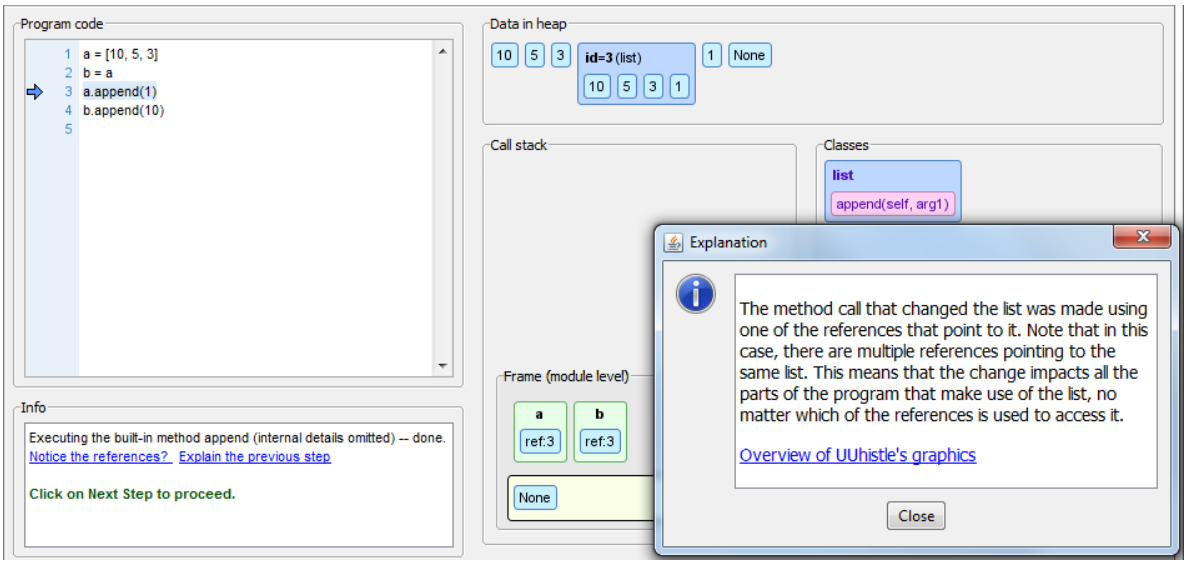


Figure 9.6: The Info box leads to further reading. This works in an animation (shown) and in VPS.

Bibliography

P. Bayman and R. E. Mayer. A diagnosis of beginning programmers' misconceptions of BASIC programming statements. *Communications of the ACM*, 26(9):677–679, 1983. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/358172.358408.

M. Ben-Ari. Program Visualization in Theory and Practice. *Informatik / Informatique, Special Issue on Visualization of Software*, pages 8–11, 2001.

B. du Boulay. Some difficulties of learning to program. *Journal of Educational Computing Research*, 2 (1):57–73, 1986.

T. Götschi, I. Sanders, and V. Galpin. Mental models of recursion. *SIGCSE Bulletin*, 35(1):346–350, 2003.

S. Holland, R. Griffiths, and M. Woodman. Avoiding object misconceptions. *SIGCSE Bulletin*, 29(1): 131–134, 1997. ISSN 0097-8418. doi: http://doi.acm.org/10.1145/268085.268132.

N. Myller, R. Bednarik, E. Sutinen, and M. Ben-Ari. Extending the Engagement Taxonomy: Software Visualization and Collaborative Learning. *Trans. Comput. Educ.*, 9(1):1–27, 2009. doi: http://doi.acm.org/10.1145/1513593.1513600.

D. C. Phillips, editor. *Constructivism in education: Opinions and second opinions on controversial issues*. The National Society For The Study Of Education, 2000.

N. Ragonis and M. Ben-Ari. A long-term investigation of the comprehension of OOP concepts by novices. *Computer Science Education*, 15(3):203 – 221, 2005.

J. Sorva and T. Sirkiä. UUhistle – A Program Visualization Tool for Introductory Programming Education (web site). `http://www.UUhistle.org/`, 2010.

10 Automated Visual Feedback from Programming Assignments

Petri Ihantola, Ville Karavirta, Otto Seppälä
Aalto University
Department of Computer Science and Engineering
Finland

{petri.ihantola, ville.karavirta, otto.seppala}@aalto.fi

10.1 Introduction

Most modern assessment platforms for programming assignments are web-oriented and used with a browser (Douce et al., 2005). Students upload their work to the system and get (almost) immediate feedback from different aspects of their program, e.g. correctness, test coverage, style, etc (Ala-Mutka, 2005). We feel that the browser platform, however, is not used up to its full potential. Using HTML and JavaScript in the feedback itself could open new possibilities. In this paper, we explore ways for offering visual feedback in existing systems for automated assessment of programming assignments without radical changes to the assessment platforms. We also present initial results of effectiveness and some results from an attitude survey.

The use of visualizations in automated assessment of correctness of programming assignments is rare. Students get textual feedback from the functionality of their (textual) programs. Yet, as summarized by Helminen and Malmi (2010), visualizations are widely used in programming education, e.g. in programming learning environments such as Scratch, Jeliot, BlueJ and GreenFoot, and programming microworlds such as Karel.

The difficulty of creating visual feedback is probably the main reason of not using visualizations in automated assessment. The assessment of the student's solution is typically based on either unit tests or output comparison, both of which traditionally have a textual outcome - an assertion error message or a description of a difference between lines of text. Most assessment platforms also aren't very flexible in this sense and do not provide any easy or well documented way of going beyond unformatted text output.

When the textual output is used in a web environment we are however given a lot more options. Our key idea is: we can make feedback visual or even interactive by embedding HTML and JavaScript in the test descriptions of (JUnit) tests. This approach enables the use of visualizations in existing assessment systems without (or with minimal) modifications to the systems.

We explain the technical part of how to construct visual feedback in Section 2. In Section 3, we present initial results of an evaluation study done on a programming course comparing general visualizations with custom visualizations tailored for the domain of each exercise as well as textual feedback. Our findings and related research are discussed in Section 4 and Section 5 concludes our work.

10.2 Introducing Visual Feedback to Existing Assessment Platforms

Assume that we had a platform capable of providing feedback from student solutions that could include or even dynamically create visualizations that make the feedback more effective. For example, instead of using textual feedback to describe what is tested, we could illustrate that test case with visualizations. Complex data such as data structures constructed during the execution of programs could be shown as diagrams.

Our goal is to change the type of feedback delivered on an existing "text-based" platform. On high level, there are at least three different approaches to perform the change: *modifying the platform*, *writing a plugin* or *modifying the exercises/tests*.

All three approaches have their strengths and drawbacks. Modifying the platform requires deep knowledge of the platform architecture and implementation. Similarly, a familiarity with the plugin architecture and APIs is needed to implement the plugin approach. On the other hand, these two are the most flexible

approaches providing control over almost everything. Modifying the tests has the lowest learning curve, especially since the teacher typically writes the tests. On the downside, the approach is the most limited. Tests are executed in a sandbox and only the test results and test descriptions (e.g. names of tests and assert strings) are passed out from the box. Despite the challenges, we decided to explore what can be achieved by modifying only the tests.

10.2.1 Modifying Tests to Enable Visual Feedback

One approach to testing student solutions is based on unit testing. An exercise has multiple tests, and each test has multiple assertions that have to pass. When an assertion fails, a string describing that specific assertion is used as feedback. The fact that in an online environment the test results are shown inside a browser creates new possibilities not widely used. Feedback (e.g. assert strings) can contain HTML and JavaScript executed on client-side. For security and convenience to the teacher, web applications are expected to escape anything that could be misinterpreted as HTML. As embedding HTML is the very thing we want, the assessment platform should not perform escaping or should provide a way to selectively prevent it.

Drawing images manually is laborious and limited. For example, we can draw visualizations related to the model solution, but cannot anticipate and draw visualizations of all the possible data structure that the students' programs will produce. We wanted to explore how to construct such visualizations dynamically. There are at least two approaches to do this:

- HTML, CSS, and JavaScript can be used to draw the image entirely in the browser.
- Image tag with a dynamically constructed url pointing to an external web service (e.g. Google Chart API[1]) constructing the image can also be used.

We used both of these approaches – HTML and CSS to construct exercise specific visualizations and Google Chart API to draw generic object graphs. The following two subsections describe these.

Exercise Specific Visualizations

Exercise specific visualizations are, as the name suggests, written for a single exercise and they visualize the relevant parts of the exercise. We had exercises dealing with a video track editor and a chess game loader. In both cases we could visualize the state of the student's program and the expected state using a specialized visualization. For the chess exercise, we first tried a specialized JavaScript library to draw the board, but ended up doing it with HTML and CSS, as illustrated in Figure 10.1. In the video track exercise we wrote our own visualizations with HTML and CSS.

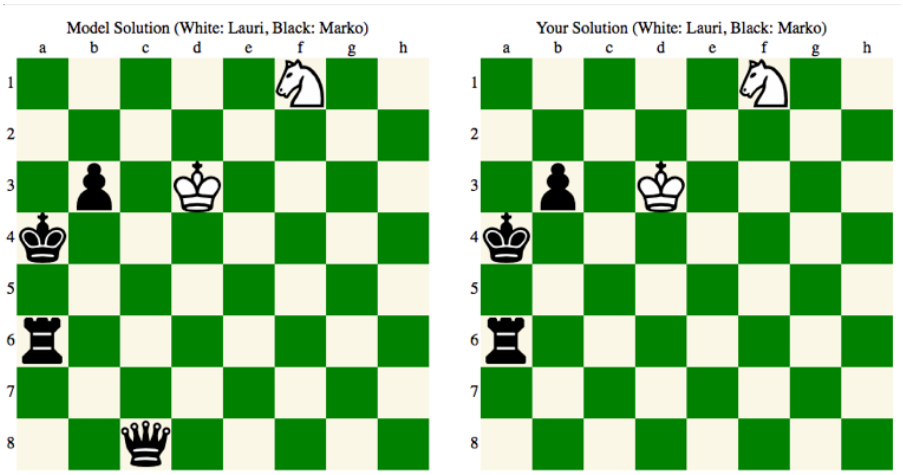


Figure 10.1: An example of exercise specific feedback from an assignment where students should construct an object graph based on a chess game state read from a text file.

[1] `http://code.google.com/apis/chart/`

The good thing about this approach is the ease of selecting only the relevant data from the assignment to be visualized and customizing presentations for the exercise. However, this does require some effort by the exercise designer implementing the visualizations in the tests. A more general way to give visual feedback is presented next.

Generic Visualizations based on Object Graphs

Automated assessment of a programming assignment is often based on comparing a student's solution against the model solution. This comparison can be performed by comparing textual outputs or object structures. In this work, we visualize the differences between object graphs.

Hamer (2004) has implemented a small tool called LJV to visualize Java object graphs. The tool uses Graphviz[2] as an intermediate language in the construction of visualizations. LJV constructs an object graph by traversing from the root node and passing the graph to a locally installed Graphviz. We wanted to push this idea further by removing the need of installing Graphviz. Unfortunately, we found the source code of LJV[3] only after we had implemented our own solution and were thus not able to build on that.

The Google Chart API is a RESTful web service capable of visualizing DOT-files (Graphviz's format). We can put a dot-description of a graph to the url of an image tag and get a rendered graph back from the service. To provide an example, `https://chart.googleapis.com/chart?cht=gv&chl=digraph{A->B->C->A}` results in an image of a cyclic graph with three nodes. We can pass such feedback as HTML and students' browsers will call the Chart API to render the feedback.

In our early experiments we found it hard to compare two large graphs. To overcome this problem, we limited information and provided visual hints to locate the differences between the graphs.

Not all parts of object graphs are relevant to the assessment. For example, in an assignment where students are asked to build a binary search tree, only the references pointing to the left child, the right child, and the data are relevant. If a student stores some extra information to a node (e.g. for debugging purposes), this information is not relevant. Thus, a way to configure how object graphs are extracted is needed. LJV has a configuration interface to do similar tasks. In our prototype, we offer something similar. Teachers can control the construction of graphs by implementing one method for each object type. It is also possible to control if selected objects (e.g. Strings) should not be visualized as independent objects (i.e. boxes) but as fake primitives inside the object referring to them. We found this necessary in order to keep our images readable.

Visualizing Differences Between Object Graphs

Before visualizing differences between object graphs we need to know how the two graphs differ, or alternatively, which parts of the graphs match. We decided to draw both graphs and use colors to highlight differences between nodes and references of the graphs – green to mark that an element has a matching pair (it is correct), red to mark that there is something wrong, and black to mark that those parts of a graph have not been compared because of earlier problems.

Comparison starts with root nodes from both graphs and proceeds in breadth first manner to search for unmatching nodes. Visualizations to highlight the differences between the two graphs are constructed during the comparison algorithm. Nodes in both graphs are compared based on:

1. **The data stored inside the nodes**. If the data are equal, both nodes are colored green and otherwise nodes are colored red and the comparison stops.

2. **The lists of references going out from the nodes**. Labels of these references can be empty in which case only the length of the lists is important or the labels can have semantic meaning, e.g. names of the references (e.g. right, left, data). Comparison of lists is based on the difference of the lists of labels of outgoing edges. Difference is calculated with the java-diff-utils library[4]. Comparing two lists, A and B, produces a patch describing how to modify B to make it equal with A. A patch is a list of deltas where each delta is either insert, edit, or delete. We calculate two patches – one from the model solution to the student's solution (P1) and one from the student's solution to the model solution (P2). Inserts in the patch from A to B are deletes in the patch from B to A and vice versa. To visualize a patch (i.e. a difference in outgoing edges of a node), we highlight deletes

[2] `http://www.graphviz.org/`

[3] `http://www.cs.auckland.ac.nz/~j-hamer/LJV.java`

[4] `http://code.google.com/p/java-diff-utils/`

and edits of P1 in the graph of the model solution and deletes and edits of P2 in the graph of student's solution. The model answer can also contain references colored in red. This means that those references are different in or missing from the student's solution. An example from such case is provided in Figure 10.2.

The comparison of lists of references going out from the nodes can be illustrated with the following two examples. First, A = [1, 2, 3, 4, 5] and B = [1, 2 ,5] is a simple example leading to highlighting 3 and 4 from A. Those need to be deleted from A to get B. A case where patches P1 and P2 both have deletes is more complicated. For example, A = [1, 2, 3, 4] and B = [1, 2 ,5] results in highlighting 3 and 4 in A and 5 in B.

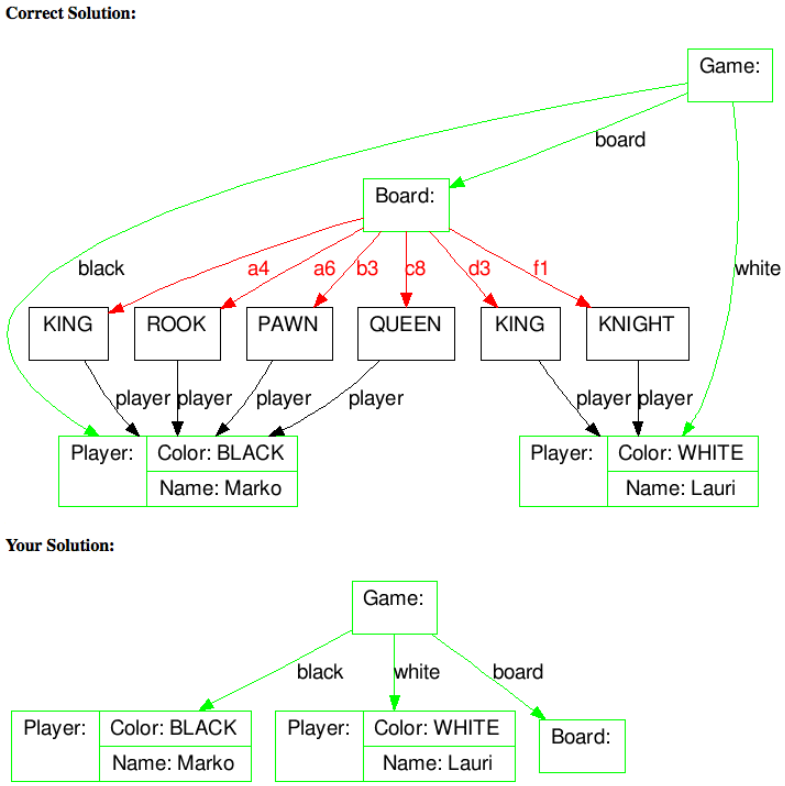


Figure 10.2: An example of generic feedback related to chess board construction exercise.

Finally, layouts of the graphs need to be similar to maintain the readability. Chart API handles the layout and if the two graphs are not identical, matching nodes can be in different positions. Based on our experiment, it is often enough if the order of references from a common node is controlled. This can be controlled with a single setting in the graph description.

10.3 Evaluation Study

10.3.1 Research Setup

We had our research setup on Intermediate Programming in Java course at Aalto University in Spring 2011. The course teaches object oriented programming in Java and is worth 6 ECTS-credits. The course has a 5 ECTS-credits CS1 course as a prerequisite. Automatic assessment is used on both courses. The exercises count for 30 percent of the course grade. All exercises were assessed with Web-CAT (Edwards, 2003). The number of resubmissions was not limited. Support groups where students could come and ask for help were provided but the attendance was not mandatory.

Students participating in the experiment were divided randomly into three groups – Groups A, B and C. First exercise round was designed to be the pretest ensuring that the groups are not different. Therefore, all groups got textual feedback from this round. In the second round, group A got textual feedback, B received specialized visualizations, and C got object graph based generic feedback. In the

Table 10.1: Feedback versions of the groups on different assignments on the course. Numbers in parentheses are the number of students who got visual feedback from the number of students who submitted a solution.

Assignment	Group A	Group B	Group C
1.1	textual (71)	textual (63)	textual (66)
1.2	textual (67)	textual (61)	textual (63)
1.3	textual (62)	textual (54)	textual (56)
2.2	textual (65)	specific visualization (9 / 57)	generic visualization (16 / 61)
2.3	textual (49)	specific visualization (32 / 44)	generic visualization (43 / 55)
3.1	generic visualization (21 / 48)	specific visualization (18 / 53)	textual (57)
3.2	generic visualization (21 / 43)	specific visualization (14 / 43)	textual (51)

third and last round, feedback types between groups A and C were swapped. The rationale behind this setup is that we were most interested to compare textual feedback against our generic, object graph based visualizations. Grading criterion between the groups were equal.

We gave visual feedback based on the comparison of object graphs. This implies that students had to be able to construct something comparable to see visualizations. If a student had a bug in his or her program that caused an exception, visual feedback was not possible. In this case, the feedback was the same as for the group getting textual feedback.

To compare the groups, we collected submission times and the grades/feedback from Web-CAT. After all the exercises were closed, we asked students to take a voluntary questionnaire about their attitudes and opinions related to visual feedback.

10.3.2 Preliminary Results

Submission Data

The number of students from each group who submitted something to the exercises is summarized in Table 10.1. In exercises where a group got visual feedback, two numbers are given. The first number is how many students got visualizations and the second the total number of students. From this data, exercise 2.3 is the most interesting. From the other assignments, majority of the students got traditional feedback making the comparison between groups difficult.

For each student and each exercise we first identified the best submission. This is the submission where the product of the three factors of grading (i.e. percentage of teacher's test passing, test coverage of student's own tests, and the percentage of student's own tests passing) is the highest. From this submission we retrieved the factors of the grade.

For each exercise and for each of the three factors, neither Kruskal-Wallis nor one way ANOVA were able to reject the null hypothesis that the groups are similar ($p>0.05$).

Although no differences were found, details of distributions in assignments with visual feedback are of interest. In all exercises and with every feedback type, only few outliers did not get full 100% from their own tests. Boxplots describing the distributions of teacher's tests passing and coverage of students' own tests are in Figure 10.3. What we can interpret from the data is that exercises 2.3 and 3.2 were the most difficult. The fact that students often write bad tests from where they get good scores has also been reported by Aaltonen et al. (2010).

As most students got full points from many of the assignments, it is more interesting to analyze how they got to this situation. Thus, for each student and each assignment, the total number of submissions was calculated. Again, no statistically significant differences were found between the groups ($p>0.05$).

Questionnaire

An interesting question is whether they would have liked to get more textual or more visual feedback. Results of this question are shown in Table 10.2 divided between students who got visual feedback and who did not. It can clearly be seen[5] that those who got visual feedback are more positive towards it, whereas students who only got textual feedback would prefer it in the future as well. How much did the students use different types of feedback was also triangulated with multiple questions. The answer is that most students use all the time all the feedback available.

[5] In fact, the difference between the groups is statistically significant (Kruskal-Wallis test, $p<0.05$).

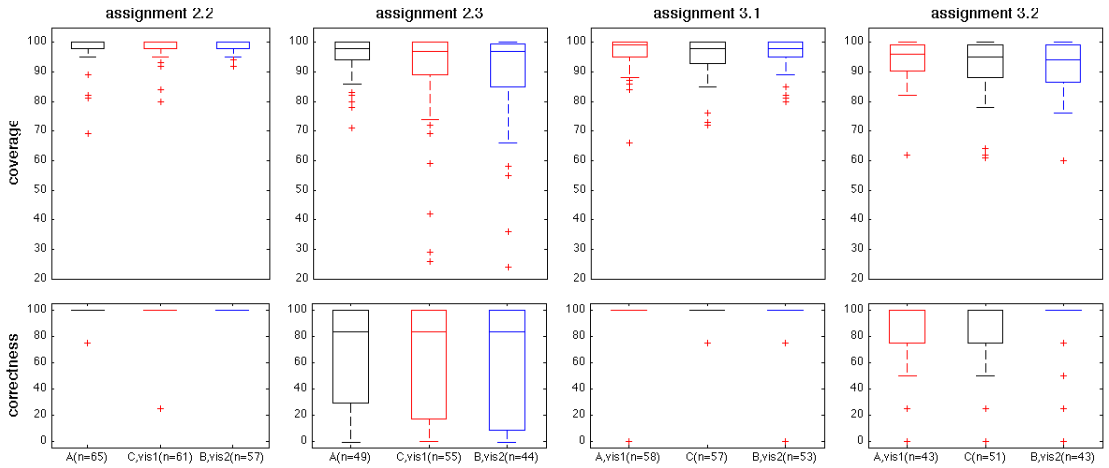


Figure 10.3: Boxplots related the performance metrics of different groups in the exercises where some students got visual feedback. Groups labeled with vis1 and vis2 got generic graph visualizations and exercise specific visualizations, respectively.

Free comments about visual feedback included positive things like: more efficient to deduce where the problem is in the code, more exact, more compact, and _always_ nicer to look at. Negative things mentioned were: the visualization was a mess, visual feedback was confusing but helpful once one understood what it meant, and visual feedback should include more verbal description how it should be interpreted. Two students commented that verbal feedback is in most cases more useful, unless the exercise deals with a problem naturally visualized (like the chessboard exercise). Furthermore, one comment mentioned verbal feedback to be more clear while another noted that verbal feedback requires knowledge about the terminology and understanding of the logic of the creator of the exercise.

10.4 Discussion and Related Research

Ma et al. (2009) have observed students to perform better with visualizations designed specifically to visualize object references are used when compared to generic visualizations of Jeliot 3. Our data does not provide statistically significant support to this. Partially this can be because in many exercises, students solved assignments without seeing visualizations. This means that after their programs managed to run without exceptions, the data structures constructed were also correct. Exercise 2.3 was the most difficult and also the only one where the majority of the students needed/got visual feedback. In this exercise, students getting textual feedback performed the best, graph based being the second and specialized feedback third (see Figure 10.3). Differences, however, were not significant.

The course we used in our study has used almost the same assignments for several years. Textual feedback from these exercises has also matured. In order to develop textual feedback of the same quality, a substantial amount of work is needed. Accordingly, designing good custom visualization can also take time. Thus, we argue that constructing graph based visual feedback is the most effortless. Therefore, visual feedback not performing significantly worse than mature textual feedback is a good result.

Visualizations seem to be useful only if assignments are difficult enough or with certain types of assignments. Programs should not fail before they return something to be compared and that something should difficult enough to construct. To better understand when to use visual feedback, different types of errors in different types of programming assignments needs to be studied more.

Table 10.2: Attitude towards visual or textual feedback. Students are divided between those who got visual feedback and those who did not.

Visual Feedback	Only Visual	Mostly Visual	Mostly Textual	Only Textual
Yes	0 (0%)	28 (67%)	13 (31%)	1 (2%)
No	1 (5%)	6 (32%)	10 (53%)	2 (11%)

Ben-Bassat Levy et al. (2003) have pointed out that visualizations can help by providing a common vocabulary and model to better understand program execution. Verbal feedback in our survey data supports this. Despite the benefits of visualization, Ben-Bassat Levy et al. conclude that *"the interpretation of the animation itself is non-trivial and must be explicitly taught."* Based on the feedback we got from our students, the same applies for static visualizations as students requested feedback images to be better explained.

The way visualizations are used is often more important than what the visualizations contain (Hundhausen et al., 2002). Introducing visualizations through HTML/JavaScript has the potential of making feedback interactive. For example, feedback could contain interactive algorithm animations or questions directing the construction of visual feedback. In future, we hope to see JavaScript libraries or web services supporting construction of these kinds of feedback. Our use of the Chart API to draw object graphs is a start in the right direction.

Security of assessment platforms needs to be improved in the future (Ihantola et al., 2010). This is emphasized when the system allows educators more freedom over the feedback creation process. Reducing responsibilities of the assessment server by externalizing such execution should help securing the server. While teachers should be allowed more control over the feedback markup, assessment systems should treat data from students' programs with care. In many assessment platforms, submissions are viewed online. Including JavaScript in the code of a submission is a possible attack vector. In addition, teachers should be aware what data they pass to external services.

10.5 Conclusions

Embedding HTML into feedback provides a flexible approach to introduce more visual feedback to current automated assessment systems used to grade programming assignments. Benefits of our approach, compared to applets or standalone visualization software, are that the approach is lightweight, it can be used with assessment tools already in place, and assignments designed with such feedback are relatively easy to port from one assessment platform to another – assuming both platforms are used with a browser.

For authors of assessment systems, supporting this is straightforward. The feedback should not be HTML escaped or there should be an option to selectively prevent escaping.

For the teacher creating the test, generic object graph visualization can give a more effortless way compared to exercise specific visualizations or textual feedback. The preliminary results of our evaluation of such feedback showed no (statistically significant) differences in student's results between any of the feedback types.

Bibliography

Kalle Aaltonen, Petri Ihantola, and Otto Seppälä. Mutation analysis vs. code coverage in automated assessment of students' testing skills. In *OOPSLA companion*, SPLASH '10, pages 153–160. ACM, 2010.

Kirsti Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102, 2005.

Ronit Ben-Bassat Levy, Mordechai Ben-Ari, and Pekka A. Uronen. The Jeliot 2000 program animation system. *Computers & Education*, 40(1):1–15, 2003.

Christopher Douce, David Livingstone, and James Orwell. Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput.*, 5(3):1–13, 2005. ISSN 1531-4278. doi: http://doi.acm.org/10.1145/1163405.1163409.

Stephen H. Edwards. Rethinking computer science education from a test-first perspective. In *OOPSLA Companion*, pages 148–155. ACM, 2003. ISBN 1-58113-751-6. doi: http://doi.acm.org/10.1145/949344.949390.

John Hamer. A lightweight visualizer for java. In *Proceedings of Third Program Visualization Workshop*, pages 54–61, 2004.

Juha Helminen and Lauri Malmi. Jype - a program visualization and programming exercise tool for python. In *Proceedings of SOFTVIS '10*, pages 153–162. ACM, 2010.

Christopher D. Hundhausen, Sarah A. Douglas, and John T. Stasko. A meta-study of algorithm visualization effectiveness. *JVLC*, 13(3):259–290, June 2002.

Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of Koli Calling '10*, pages 86–93. ACM, 2010.

Linxiao Ma, John D. Ferguson, Marc Roper, Isla Ross, and Murray Wood. Using cognitive conflict and Jeliot visualisations to improve mental models. In *Proceedings of ITiCSE '09:*, 2009.

11 Truly Interactive Textbooks for Computer Science Education

Clifford A. Shaffer*, Thomas L. Naps**, and Eric Fouh*
*Department of Computer Science
Virginia Tech

**Department of Computer Science
University of Wisconsin, Oshkosh

shaffer@cs.vt.edu, naps@uwosh.edu, efouh@vt.edu

11.1 Introduction

The dream of an electronic textbook has been actively pursued for at least two decades. Goals include (i) improving exposition through a richer collection of technologies than are available through print textbooks, and (ii) increase student engagement with the material, in order to get them to learn at a higher level in Bloom's taxonomy (Naps et al., 2002). Instead of merely viewing material, we can hope to use frequent assessment (by asking questions) to get them responding, and through interactive activities get them to change and construct virtual artifacts. We will use the term *hypertextbook* to refer to an electronic textbook that integrates interactive exercises and assessment. See (Rössling et al., 2006; Ross and Grinder, 2002) for background on efforts to define and implement the hypertextbook.

In this paper, we discuss our plans to create a hypertextbook for a complete semester course in Data Structures at the Sophomore level. Data Structures and Algorithms as a topic can particularly benefit from the use of advanced technology to aid explanation of the dynamic processes that make up the essence of an algorithm, and which can be difficult to convey using words and images. Therefore, our particular focus is on the use of algorithm visualization (Shaffer et al., 2010; Naps et al., 2002) as a means both to deliver the necessary dynamic exposition, and to increase student interaction with the material. We will discuss how research results indicating the value of AVs combined with a lack of progress in uptake of AVs in actual courses leads us to the conclusion that a complete semester-long course package is the right way to go. We describe our plans for implementation, including a discussion of relevant technology for the project.

11.2 Motivation

Periodic surveys on both instructor interest in AVs and their level of use have been conducted among attendees of education conferences and listservs for more than a decade. Naps et al. (2002) report on three surveys conducted in 2000. Collectively, they indicate a strong positive view in favor of AV use (over 90%). However, only about 10% of respondents at that time indicated frequent use of AVs in data structures courses at their institutions, while about half to three quarters indicate occasional use of AVs in such classes.

At SIGCSE'10 (held in Milwaukee during March 2010) we conducted a new survey to determine instructor attitudes toward AVs and their use in the classroom. For details, see Shaffer et al. (2011). The survey was designed in the spirit of the 2000 surveys, and the findings are consistent with those earlier results. In 2010, 41 of 43 respondents agreed or strongly agreed that AVs can help learners learn computing science, with two neutral. However, just over half had used AV in a class within the past two years. It is not clear that all of the "yes" answers from the present survey refer to what we might consider to be "AVs." But they probably all refer to some sort of dynamic software visualization run on a computer, as was the case in the 2000 survey. We did not ask about frequency of use of AVs in class.

Our third survey question asked what respondents see as the greatest impediments to using AVs in courses. Roughly half the responses relate to finding suitable AVs to use. We can hope that the presence of online resources such as the AlgoViz Portal (`http://algoviz.org`) will reduce this information gap,

by making it easy for instructors to locate quality AVs. However, about half of the responses relate to issues with integrating AVs into courses. These results are roughly the same as reported in Naps et al. (2002, 2003) and Rössling et al. (2006). Such issues are much harder to deal with, and are representative of well-known problems for adoption of educational technology in general (Hew and Brush, 2007). While it is easy to give students pointers to AVs as supplemental material, it is much harder to integrate AVs into lectures, labs, and assignments. Respondants cite lack of time to make course changes, lack of time within the course to spend additional time on a given topic, and inconsistencies between the course textbook and the AV.

We hope that community support provided by the AlgoViz portal can help. The model embraced by AlgoViz is to allow the community to add value beyond information embodied in a simple catalog of AVs. This value comes as a byproduct of direct communication between community members. While forums are one obvious method for this, there is a deeper communication involving community ratings and recommendations for content entries and sharing experiences on how to make the best use of content. A major concern for instructors is deciding whether a given educational resource is of good quality, and how to use it. Therefore, user feedback on resources is as important as the resources themselves. AlgoViz provides the *field report*, which gives a convenient way for instructors who have used an AV in a class to share their experiences. Field reports are designed to supplement the evaluation data in the AV Catalog, since AV ratings in the catalog are not necessarily based on real-world classroom experiences. A field report can thus provide empirical evidence to strengthen or qualify a recommendation.

The survey results lead us to the conclusion that it is easier to integrate a complete block of instruction (either a complete topic or even a semester course) than it is to fit a piece of instruction such as an AV into existing presentation on that topic. Instructors are used to the concept of adopting a new textbook for new courses that they teach, and often welcome lecture notes and other class support artifacts. Even for courses taught previously, instructors will adopt new textbooks and new presentations for various topics in the course. A key aspect is that adopting a new chunk of content allows the instructor to completely replace or populate a block of course time, as opposed to squeezing new content or presentation techniques on top of existing material. In the past, most AVs developers have implicitly taken the approach that their AV will be integrated into existing presentations. A typical example might be an AV presenting a Quicksort algorithm, with no tutorial explanation of the algorithm. The idea seems to be that this visualization can be slipped into lecture or used by students to supplement the textbook for self-study. But this approach has led to the problems noted above. In contrast, a complete unit of instruction (including AVs) can more easily replace an unsatisfactory existing presentation of the topic.

Important technological factors contribute to making AVs easier to use in the classroom than in prior years. More AVs are available than ever before (Shaffer et al., 2010), backed up by improved research studies and improved access both through general Internet search and at the AlgoViz Portal. Increased access to the Internet by students and instructors, both in and out of the classroom, makes AV use more practical. For example, at Virginia Tech, while it has been "possible" to project Internet material from a computer in class for over a decade, it has only been in the past couple of years that such access has become both ubiquitous and reliable in all of our classrooms. This makes a huge difference in the confidence of mainstream instructors for using such technology, in contrast to early adopters (Hew and Brush, 2007).

Another potential factor in favor of the use of AVs and hypertextbooks is ubiquitous availability of laptops and mobile devices. For example, all Engineering majors at Virginia Tech are required to own a tablet PC, and most also have mobile devices including smartphones, ebook readers, or iPads. However, there is a downside to the non-PC devices, in that they have various technology limits on how eTextbook content can be provided. So while there is ubiquitous access to the Internet among our target audience, there are still limits. For example, Java Applets cannot be displayed on most such devices.

11.3 Prior Work

The bulk of effort by CS educators involved with interactive AVs has focused on development of the AV technologies themselves and at best small, focused hypertextbook modules that incorporate AVs into a particular topic or small set of topics within a course. For example, Grinder et al. (2002) report on a set of web pages integrated with applets that were used in parts of a theory of computation course. Many instructors who teach theory now use Susan Rodger's JFLAP (Gramond and Rodger, 1999) throughout much of their course and Rodger has published a hard copy guide to using JFLAP (Rodger and Finley, 2006). In the preface Rodger warns the reader "our book assumes that the reader has read briefly about these topics first in an automata theory textbook or a compiler textbook".

Ross (2008) describes efforts at building a Perl-based infrastructure for creating hypertextbooks. The infrastructure relies on the Dreamweaver toolkit. Only a few chapters (three out of seven) of a theory of computing and a biology book has been produced using this technology. The system requirements for these hypertextbooks also hinder their wide adoption due to browser restrictions, use of Java applets, and specific screen resolutions.

Rößling and Vellaramkalayil (2009) report on a technology that integrates AV into Moodle-based lessons, but the emphasis of that report was again on a technology that would support visualization-based hypertextbooks in Moodle, and to our knowledge little progress has been made on the actual writing of such a textbook.

Titterton et al. (2010) have used a lab-centric mode of instruction for introductory CS courses at UC-Berkeley. Their current work is being done in Moodle and uses a small amount of AV along with many other "check point" exercises that students must complete as they progress through a set of material presented in Moodle. It is built upon an earlier technology called UC-WISE that was developed and used exclusively at UC-Berkeley. Their materials are designed for introductory CS courses that aims mostly at developing students' programming skills. Hence they make only limited use of AVs. Alharbi et al. (2010) are using eXe, an open source authoring system for academic-related web content using XML and HTML (`http://www.exelearning.org/`). Their efforts intend to help teaching Operating Systems using visualization. Learners can interact with the AVs, and can take quizzes and tests online. They used the Sharable Content Object Reference Model (SCORM: `http://www.adlnet.gov/`) to support integration of digital teaching materials with a CMS (Moodle in their case).

Another effort to integrate AVs into hypertext is being performed by Karavirta (2009). His solution is based on HTML and JavaScript, allowing the hypertextbook to be viewed in any browser without additional plug ins. The learner can interact with the animation and draw annotations on it, but the current system does not store the annotation, nor does it support quizzes and tests. Their earlier work on TRAKLA2 (Malmi and Korhonen, 2008) includes a tutorial on heaps that is integrated with AVs. Animal (Rößling et al., 2000) and JHAVÉ (Naps, 2005) are AV development systems that include tutorial descriptions for some single AVs. Crescenzi and Nocentini (2007) takes the novel approach of a traditional textbook whose examples and illustrations are closely tied to the AlViE AV system.

Largescale use of educational hypermedia in South Korea. provides interesting feedback about the challenges of hypertextbooks. Kim and Jung (2010) identify usability, portability, interactivity, and feedback as major elements to consider while designing such systems. Learners should be able to ask questions and receive help, as well as control, manipulate, search and browse hypertextbook content. They also advocate for the development of models to support group collaboration.

There exists no complete electronic textbook, tightly integrated with AVs, that could be used as the primary learning resource in a semester-long computer science course. This is perhaps surprising because Marc Brown's groundbreaking dissertation on AV ((Brown, 1988)) issued the following caution:

> *Much of the success of the BALSA system at Brown [at the time Brown's thesis was written] is due to the tight integration of its development with the development of a textbook and curriculum for a particular course. BALSA was more than a resource for that course – the course was rendered in software in the BALSA system.*

Why have CS educators not heeded Brown and authored hypertextbooks? We suspect the answer is something known to anyone who has written a textbook or an AV: it consumes huge amounts of time. While writing a textbook is a big job, writing and associated set of AVs and the assessment support is a far bigger job yet.

11.4 A Case Study

In this section we briefly describe an online hashing tutorial and a study conducted to determine its pedagogical efficacy. The results have helped us to make progress toward defining the requirements for a more comprehensive electronic textbook. Note that this approach is similar in spirit to the TRAKLA2 heaps tutorial (`http://svg.cs.hut.fi/heaptutorial`).

In 2008 and repeated again in 2009, students in separate sections of a sophomore level course on data structures and algorithms at Virginia Tech were taught about hashing. One section was given standard lecture and textbook for one week, equivalent to what had been done previously in the class. The other section spent the class time working through an online tutorial combined with algorithm visualizations to present the same material. The tutorial used text content taken from the course textbook, so that it was an exact match to the material being presented in the control section. However, the online

tutorial heavily supplemented this text with algorithm visualizations. The tutorial can be found online at `http://research.cs.vt.edu/AVresearch/hashing`.

In each of the trials, the two sections were given a quiz on hashing at the conclusion of the week of instruction. The results were positive: An analysis of variance shows a significant difference between the two treatment groups ($F(1, 118) = 4.37$, $p < 0.05$), with students completing the tutorial averaging higher quiz scores than those who were given the lecture and textbook content. However, the difference in the means is 6.2 percentage points, or approximately one third of one standard deviation. These results indicate that not only can an online tutorial be as effective as lecture (which has important implications for distance learning), but that providing proper interactivity allows computerized instruction to be better than lecture-based (passive) instruction. However, there is at least one major consideration that might influence the results of this particular study: How much impact did the controlling structure of coming to class and doing the tutorial in "lab" setting have on the results? The outcome could be quite different for a student just reading the material and working through the visualizations on their own, where self-discipline might well not be sufficient to provide the necessary amount of time and attention. Likewise, the controlled environment of attending lecture before reading the textbook on one's own is also likely to have a major difference compared to just reading the book on one's own.

We hypothesize that, if we want to have a viable self-contained electronic textbook that supports self-study of the material, the system needs to build in some equivalent to the controlled pacing and feedback that is encountered when attending classes or labs. This means that assessment and an objective measure of "progress" through the material needs to be an integral part of the system (whether purely self assessment or with results submitted to an instructor). This means a much tighter integration of material, interactive exercises, and assessment than is provided by a system as simple as the Hashing Tutorial.

11.5 Implementation Principles for an Electronic Textbook

Based on our interpretation of the results of the 2010 survey (Shaffer et al., 2011), our experiences with the Virginia Tech Hashing Tutorial, and the continuous improvements in online technologies that we observe, we think that the time is ripe to create an entire online textbook for an undergraduate course on Data Structures and Algorithms. Depending on the exact topics to be covered, this could be beneficial at any level from CS2 through senior algorithms.

When considering the full breadth of content that would be contained in a complete course textbook on the subject, we conclude that three distinct forms of content presentation are desirable. First, no matter how dynamic and interactive the topic, text and images as are now found in typical textbooks continue to have their place as part of the exposition. Some content simply is not visual or dynamic and so is efficiently transferred via words and images.

Second, some content is essentially expository (i.e., at that point in the presentation there is no need for student constructive interaction), but the content is about dynamic processes or conducive to visual presentation. This includes most algorithm descriptions, such as how a particular sorting algorithm works. Since initial presentation does not involve exploration or decision making, or demonstration of proficiency, the prime concern is what techniques provide the clearest explanation. This could best be handled by a presentation that relies heavily on diagrams and simple animation, with pacing controlled by the the reader. In essence, an animated slide show is adequate for such presentations.

Third, there many instances that from student interaction. This includes things as simple as probing a calculation, (e.g., trying different inputs to a simple simulation or calculation). An example is the famous Birthday Problem: How many people need to be in a room before the odds are greater than even that two share a birthday? Another need for interaction comes with demonstrating proficiency with an algorithm, such as the interactive exercises for tree insertions that are part of TRAKLA2 (Malmi and Korhonen, 2008; Malmi et al., 2004). These are best handled by something like a Java Applet to support user interaction and dynamic on-the-fly calculations/processing of the algorithm. Performance comparisons are also of this nature, where the student is invited to define and run built-in simulations of multiple data structures or variants to see how they perform. Binding all of this together should be a steady stream of assessment activities to make sure that students stay "on track" and to keep them engaged even during otherwise passive exposition. This can be done with simple pop-up questions (that might or might not require a successful response to continue) and end-of-section quizzes whose success might be required to demonstrate competence needed to continue to the next section.

As we progress from the first to the third of these presentation approaches, the development cost goes up greatly. Text and images can be developed relatively quickly. In contrast, it took several student-years of effort (and over two actual years) to develop the Hashing Tutorial, mainly due to the effort

involved in developing a handful of Java applets. A properly animated slideshow takes longer to create than equivalent text and images, but should be significantly faster to implement than a fully interactive exercise.

11.6 Technical Considerations

A number of technologies are available for developing the three presentation types for the electronic textbook as was envisioned in the previous section. Text and images can certainly be done with any traditional web development tool. The second component, which we characterize as an animated slide show, can be done (as the characterization suggests) using a variety of presentation tools such as MicroSoft Powerpoint, LaTeX's Beamer package, OpenOffice Impress, or Apple Keynote. However, while presentations can be created in these tools, the resulting presentations cannot so easily be integrated with the rest of the electronic textbook. Browsers can support some of these tools as plugins, but not universally across a range of devices. The presentations can generally be converted to PDF format, but only Adobe's Reader can actually display the animations (at least, evince and xpdf, popular alternative PDF readers do not). Nor do the PDFs well integrate with non-PDF portions of the whole.

Flash is another popular tool for developing animations, and is rich enough to support the interactive aspects of the third component of presentation as well. However, Flash requires a plugin in most browsers, and so is not compatible with devices such as the iPad.

One technology that appears to be robust enough to implement all desired dynamic and interactive components is HTML5 incorporating JavaScript. HTML5 integrates its dynamic components well with standard text and images, and easily ports between PC browsers and mobile devices. Potential concerns include ease of use for content developers (as compared to, for example, PowerPoint when developing animated slide shows), and level of penetration of the necessary browser technology. Given that alternatives such as Flash are at least as problematic in terms of a typical user having access (since they require plugins), the problem of penetration seems to be low and quickly receding, in that we can expect that college-level students tend to have access to moderately up-to-date browser technology. Thus, we currently advocate use of HTML5 technology for developing the dynamic components of the electronic textbook.

11.7 Integrated Assessment and Progress Monitoring

There is much evidence that AVs foster effective learning when presented in a way that forces the student to actively engage with the visualization instead of passively viewing it (Hundhausen et al., 2002; Naps et al., 2002). Additional engagement can be created by having the student respond to interactive questions. Although many AVs include questions, few do so in a way that allows an instructor to monitor their students' progress. More typically the student's interaction with the system produces immediate feedback to the student, but the assessment of that interaction is not recorded in a way that the instructor can access. Nor do the students' answers provide a persistent record for the student that a section has been mastered, or integrate with navigation through the content such as provided by an "intelligent tutor". Two AV systems that do support this sort of integrated assessment are TRAKLA2 (Malmi et al., 2004; Malmi and Korhonen, 2008) and JHAVÉ. (Naps, 2005).

Although the TRAKLA and JHAVÉ systems support online assessment of students' using the visualizations, they both do so in a unique, non-portable way. We seek an approach that can work with a variety of presentation as well as the electronic textbook structure itself (some but not all questions will come within AVs). One possibility is a decoupled approach to assessment, where the actual assessment process is done by following a link to a third-party site that provides the relevant series of questions. This might take place at the end of each section in the textbook. Among the factors that will have to be considered in developing this protocol are:

- Developing effective strategies for assessing student responses. Assessing multiple-choice, multiple-selection, and fill-in-the-blank questions is straightforward. But automated assessment of textual responses is clearly much more difficult.
- Support for richer types of activities specific to CS, such as small programming tasks evaluated by comparing output from the proposed solution to the answer key.
- How to represent test questions. There exist standardized question representations, such as the IMS Question and Test Interoperability specification (QTI: `http://www.imsglobal.org/question/qtiv1p2/imsqti_oviewv1p2.html`).

- How to store student responses and progress in a fashion that makes it easy for instructors to analyze their students' results.
- How to store assessments of students' work in a fashion that respects their privacy.
- How to "loosely couple" the assessment system with the electronic textbook. Because the visualizations watched by our learners will be launched out of the hypertextbook, we will need to have a mechanism for recognizing the learner's identity so as to interact with the quiz/assessment database that might be stored on a different server.

11.8 Connexions and the Creative Commons

Our final consideration is the broader context in which development of an electronic textbook should take place. Such a project is a huge undertaking. As evidence of this (besides our experiences with the extraordinary amount of time that it takes to develop high-quality AVs), consider that there exist few examples of the type of artifact that we seek to create, as stated in Section 11.3. Ideally, a broader community can be encouraged to contribute to the project, much in the style of an open source software development effort. The authors have many collaborators within the broader algorithm visualization community, and ideally we could leverage these collaborations to develop the materials. Since we envision the materials to be distributed with a GPL or Creative Commons license, intellectual property rights will be less of an issue than if a commercial publisher were involved.

The Connexions Project (`http://cnx.org`) is presently the largest collection of online textbooks developed with a creative commons license model. Connexions is more than a collection of publicly available online textbooks. They have developed a "creative commons" infrastructure that makes it easy for authors to reuse and combine pieces of textbooks, or to make their own altered version of an existing textbook. We envision developing our project in such an infrastructure to support sharing and reuse of educational chunks. There are integration concerns related to, for example, HTML5 technology or other technology for developing dynamic presentations. Integration of assessment is also of concern, since Connexions has only recently begun developing support for assessment.

We envision a multistage process to develop the hypertextbook project. The first step is to devise a complete management plan and to define the development workflow within the chosen implementation infrastructure. Next is to define a detailed "storyboard" defining a detailed layout for the entire hypertextbook, with all of the text and detailed descriptions of all places where interactive activities or presentations are desired. Third is to then begin an open development process where submissions from interested parties are provided for specific activities called for in the Storyboard under a reviewing process. If developed in this way, the hypertextbook will become "owned by" the broader community of AV developers.

Bibliography

A. Alharbi, F. Henskens, and M. Hannaford. Integrated standard environment for the teaching and learning of operating systems algorithms using visualizations. In *5th International Multi-Conference on Computing in the Global Information Technology*, pages 205–208, 2010.

M.H. Brown. *Algorithm Animation*. MIT Press, Cambridge, Massachussets, 1988.

Pierluigi Crescenzi and Carlo Nocentini. Fully integrating algorithm visualization into a CS2 course: A two-year experience. In *Proceedings of the 12th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, pages 296–300, 2007. doi: http://doi.acm.org/10.1145/1268784.1268869.

E. Gramond and S.H. Rodger. Using JFLAP to interact with theorems in automata theory. *ACM SIGCSE Bulletin*, 31(1):336–340, 1999.

M.T. Grinder, S.B. Kim, T.L. Lutey, R.J. Ross, and K.F. Walsh. Loving to learn theory: Active learning modules for the theory of computing. In *Proceedings of the 33rd ACM SIGCSE Technical Symposium on Computer Science Education*, pages 371–375, 2002.

K. Hew and T. Brush. Integrating technology into K12 teaching and learning: current knowledge gaps and recommendations for future research. *Educational Technology Research and Development*, 55: 223–252, 2007.

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

V. Karavirta. Towards seamless merging of hypertext and algorithm animation. In *Proceedings of the 5th Program Visualization Workshop*, pages 105–114, 2009.

J.H.-Y. Kim and H.-Y. Jung. South Korean digital textbook project. *Computers in the Schools*, 27(3 & 4):247–265, 2010. doi: http://dx.doi.org/10.1080/07380569.2010.523887.

L. Malmi and A. Korhonen. *Active Learning and Examination Methods in a Data Structures and Algorithms Course*, pages 210–227. Number 4821 in LNCS. Springer-Verlag, 2008.

L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, and P. Silvasti. Visual algorithm simulation exercise system with automatic assessment: Trakla2. *Informatics in Education*, 3(2):267–288, September 2004.

T.L. Naps. Jhavé: Supporting algorithm visualization. *IEEE Computer Graphics and Applications*, 25: 49 – 55, September 2005.

T.L. Naps, G. Rössling, and nine more authors. Exploring the role of visualization and engagement in computer science education. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, pages 131–152, 2002.

T.L. Naps, S. Cooper, and twelve more authors. Evaluating the educational impact of visualization. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, pages 124–136, 2003.

S.H. Rodger and T.W. Finley. *JFLAP-an interactive formal languages and automata package*. Jones & Bartlett Learning, 2006.

R.J. Ross. Hypertextbooks and a hypertextbook authoring environment. In *ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education*, page 133–137, Madrid, Spain, 2008. ACM.

R.J. Ross and M.T. Grinder. Hypertextbooks: Animated, active learning, comprehensive teaching and learning resources for the web. In S. Diehl, editor, *Software Visualization*, pages 269–284. Springer, 2002.

G. Rößling and T. Vellaramkalayil. First steps towards a visualization-based computer science hypertextbook as a Moodle module. In *Proceedings of the 5th Program Visualization Workshop*, pages 47 – 56, 2009.

G. Rössling, T. Naps, and nine more authors. Merging interactive visualizations with hypertextbooks and course management. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, pages 166–181, 2006.

Guido Rößling, Markus Schüer, and Bernd Freisleben. The ANIMAL algorithm animation tool. In *Proceedings of the 5th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, pages 37–40, 2000. doi: http://doi.acm.org/10.1145/343048.343069.

C.A. Shaffer, M.L. Cooper, A.J.D. Alon, M. Akbar, M. Stewart, S. Ponce, and S.H. Edwards. Algorithm visualization: The state of the field. *ACM Transactions on Computing Education*, 10:1–22, August 2010.

C.A. Shaffer, M. Akbar, A.J.D. Alon, M. Stewart, and S.H. Edwards. Getting algorithm visualizations into the classroom. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE'11)*, pages 129–134, 2011.

N. Titterton, C.M. Lewis, and M.J. Clancy. Experiences with lab-centric instruction. *Computer Science Education*, 20(2):79–102, 2010.

12 Towards a Hypertextbook Authoring System (HAS)

Steve Aldrich, Frances Goosey, Rance Harmon, Rockford Ross[1]
Montana State University

ross@cs.montana.edu

12.1 Introduction

This is a position paper. The authors take a twofold position in this paper:

1. Truly *widespread* infusion of visualization systems into established or emerging curricula will not happen without the availability of hypertextbooks that:
   a) replace traditional textbooks (i.e., do not require a traditional textbook in addition to a hypertextbook)
   b) require virtually no learning on the part of students or instructors to navigate (i.e., are as intuitive to use as a traditional textbook)
   c) function as standalone resources (i.e., do not require other supporting computing systems that would need to be installed and learned by instructors and students)
   d) require no *technical* effort on the part of instructors to integrate into the fabric of an existing or new course (i.e., incorporate all of the notational and other conventions needed for the course so that instructors are not burdened with technical integration issues)
   e) integrate visualization systems seamlessly into the fabric of the hypertextbook
2. A Hypertextbook Authoring System (HAS) must be developed to support the authoring of standalone hypertextbooks.

This is also a work-in-progress paper. The authors have a start on a HAS prototype, to be discussed later.

Both the twofold position and the prototype HAS are presented in this paper in the spirit of a working group, namely as "undercooked" issues to be discussed, critiqued, argued, and improved as work continues on projects that will encourage the widespread use of visualization software in mainstream curricula. However, it should especially be noted that this position should not be mistaken for a denunciation of other attempts at hypertextbook construction (e.g., creation of Vizualization based Computer Science Hypertextbooks—so called VIZCoSHes—for Learning Management Systems), but rather a complementary approach that the authors believe has perhaps the best chance of infusing curricula with visualizations and animations for teaching and learning.

12.2 Background

The third author of this paper has extensive experience in designing, implementing, deploying, and evaluating hypertextbooks. One in particular, *Biofilms: The Hypertextbook*, will serve as the basis for the discussion of a HAS in this paper. He has also worked extensively with the Desire2Learn learning management system (LMS) and been involved with a substantial Drupal content management system (CMS) project.

[1] This paper is based upon work supported by the National Science Foundation under Grant No. 0618744. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

12.2.1 Biofilms: The Hypertextbook

Biofilms: The Hypertextbook is a continuing work based originally on a grant from the National Science Foundation in the United States, and can be found at `http://www.hypertextbookshop.com/biofilmbook/working_version`. *Biofilms: The Hypertextbook* has undergone extensive evaluation that has proved its worth and acceptance by instructors and students. It contains virtually all of the features one would expect in a hypertextbook: text, videos, audio, images, and interactive visualization models. Web statistics show that it is widely used in the United States and, to a lesser extent, in other countries of the world. It is currently undergoing extensive technical revision to bring it up to the latest emerging Web standards, including HTML5 and CSS3.

Other hypertextbooks in progress include *Theory of Computing: The Hypertextbook*, and *Spinning Webs: The Hypertextbook*. A possible hypertextbook on economics is also under consideration. These are on hold at the moment as the HAS infrastructure upgrades are made.

12.2.2 Desire2Learn

The third author's classroom use of the Desire2Learn LMS has convinced him that it is too cumbersome, inflexible, and isolated to be used as the basis for a HAS or for housing a hypertextbook in general. Among the drawbacks are:

- Desire2Learn creates pages on the server side from user supplied HTML that include lots of markup that is not easy to deal with in creating one's own style.
- There appears to be no way to really make a truly flexible, standalone hypertextbook in Desire2Learn (it really wasn't designed with a hypertextbook in mind).
- Content in Desire2Learn requires that a student have password access, which normally can only happen if the student is enrolled at the university and in the class in question.
- Content is not easily shared or ported between Desire2Learn and other Learning Management Systems, so content is available only to students withDesire2Learn access in a particular class at a particular university.
- Requiring students and instructors to use Desire2Learn in order to access a hypertextbook constructed for the Desire2Learn LMS would not be an effective strategy for deploying a hypertextbook, as instructors would be reluctant to convert their course from one LMS to another; furthermore, some institutions have a policy that only one LMS chosen by the university be used for all courses.

A comparison of Desire2Learn with Moodle had not been done, but it is suspected that most of the same issues would be present in Moodle.

12.2.3 Drupal

At one point Drupal was being considered, on the advice of advanced students who had extensive experience with it, as a platform for hypertextbooks and an accompanying HAS. In the end that approach was rejected for many of the same reasons that Desire2Learn was rejected. It appears to be simply too big, cumbersome, and restrictive to allow for the agile kind of hypertextbook and HAS the authors have envisioned and implemented.

12.2.4 Other Options

Other options were also explored, such as PDF (which is getting continuously more sophisticated), DocBooks, and E-books. All were rejected, if for nothing else, for their inability to support interactive visualizations.

12.2.5 Position Summary

Experience with hypertextbook construction "from scratch" along with an analysis of some existing LMS's and CMS's have led the authors to the conclusion that hypertextbooks of the type envisioned were better implemented as a special and separate class of Web object. One obvious implication of that conclusion was the need for a HAS to support hypertextbook authoring. Some of the features found in a CMS or an LMS will surely find their way into a HAS, but the result should be a specific system with one objective—the authoring of hypertextbooks—which is a venture complex and important enough to warrant its own support infrastructure.

12.3 Current HAS Infrastructure

As the conclusions outlined above were being reached, some of the infrastructure for supporting a standalone HAS was designed and implemented.

12.3.1 Dreamweaver

Currently, the HAS is based on Adobe Dreamweaver CS5. A prospective author is provided the following skeleton infrastructure to get started with hypertextbook authoring in Dreamweaver:

- A hypertextbook file structure skeleton
- A set of Dreamweaver templates for standard page creation
- A set of external cascading style sheets that govern the layout and style of the hypertextbook
- A set of Javascript programs for managing some of the dynamic and interactive aspects of the hypertextbook
- A set of snippets (HTML code to be inserted as needed for the inclusion of images, videos, and audio files)
- Instructions on how to construct a hypertextbook within this framework

12.3.2 Web Presence

For publishing the hypertextbook, a prospective author is expected to obtain an appropriate domain name (e.g., biofilmbook.com). The author is then given an FTP account under that domain name on a server provided as part of the HAS. Once the FTP parameters to the new site have been configured in Dreamweaver, publishing to the server is done with the click of a single button in Dreamweaver.

12.3.3 Interactive Feedback Quizzes

Authors are able to construct interactive quizzes for a Javascript quiz generation program that provides feedback, and to embed these quizzes in the hypertextbook.

12.3.4 Preprocessing

Each time the hypertextbook is published to the Web by Dreamweaver, it is run through preprocessing programs developed by the HAS team. These preprocessing programs currently do the following:

- Create dropdown menus for each page in the hypertextbook based on a complete scanning of the hypertextbook. This allows an author to move, delete, or insert pages, sections, and chapters at will without concern for the integrity of navigation links.
- automatically Identify and number figures, update references to figures in the text to refer to the new figure numbers, and create HTML pages corresponding to "list of figures."

12.3.5 The Whole Package

The HAS works pretty well as it is. However, it does require learning to use Dreamweaver to some extent. One must recognize, however, that any authoring system, whether it be LaTeX or Word (think equations), will require a learning curve. That said, the spiral approach we have in mind for creating a HAS is intended to gradually increase the functionality of the HAS while simultaneously ensuring that it is functional at the time. Eventually we hope to dispense with Dreamweaver entirely, replacing it with a HAS editor and functionality specific to a HAS.

12.4 Use Statistics for *Biofilms: The Hypertextbook*

The following sequence of charts provides Web statistics for *Biofilms: The Hypertextbook* for the spring 2009, 2010, and 2011 semesters. These charts were compiled by the program Webalizer available on the hosting server for *Biofilms: The Hypertextbook*. To be sure, Web statistics vary by the statistical algorithms employed, and can be misleading. However, it is reasonable to view the number of visits as a somewhat valid metric, where *visits* is defined as a sequence of requests from a uniquely identified client that expire after a certain amount of inactivity. At a minimum, such statistics can reveal trends over time, even if the actual values of the metrics are suspect.

We chose the period from January through May of each represented year to illustrate usage trends. These months correspond to the typical spring semester period for U.S. universities.

12.4.1 Spring Semester 2009

Summary by Month

Month	Daily Avg				Monthly Totals					
	Hits	Files	Pages	Visits	Sites	KBytes	Visits	Pages	Files	Hits
May 2009	2883	2468	323	110	2948	3068133	3437	10039	76534	89387
Apr 2009	3745	2972	373	134	3354	3316609	4029	11215	89183	112360
Mar 2009	2027	1736	243	95	2359	1590336	2963	7557	53822	62846
Feb 2009	1311	1074	188	57	1148	1162650	1609	5278	30091	36712
Jan 2009	967	742	161	44	707	1028052	1372	5010	23027	29988

12.4.2 Spring Semester 2010

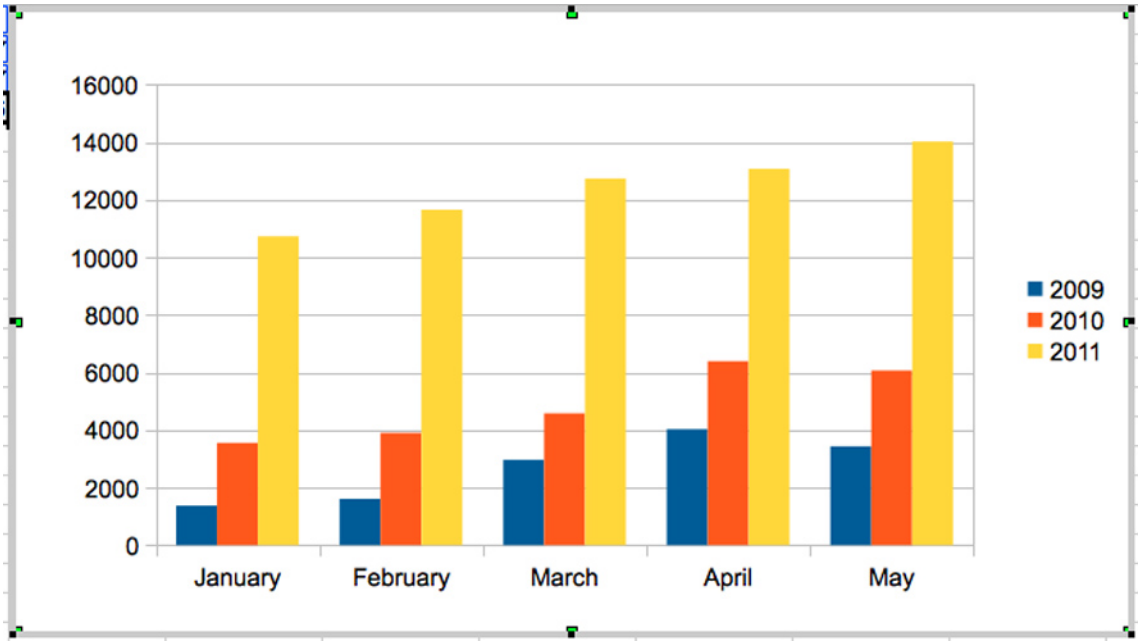
Summary by Month

Month	Daily Avg				Monthly Totals					
	Hits	Files	Pages	Visits	Sites	KBytes	Visits	Pages	Files	Hits
May 2010	5393	3226	617	195	5516	3928299	6067	19137	100011	167209
Apr 2010	6761	3900	637	213	5584	5094388	6392	19115	117013	202853
Mar 2010	3897	2339	457	147	3687	3249196	4580	14185	72535	120812
Feb 2010	3748	2154	444	139	2885	2449277	3902	12443	60313	104952
Jan 2010	2705	1640	303	114	2469	2383842	3553	9396	50856	83884

12.4.3 Spring Semester 2011

Summary by Month										
Month	Daily Avg				Monthly Totals					
	Hits	Files	Pages	Visits	Sites	KBytes	Visits	Pages	Files	Hits
May 2011	10965	7594	901	452	11124	24430930	14026	27941	235426	339928
Apr 2011	11743	8180	964	436	10995	10400910	13081	28930	245425	352311
Mar 2011	11790	7909	994	411	11249	11987675	12744	30815	245195	365500
Feb 2011	11929	7753	1075	416	10378	10023746	11657	30117	217095	334039
Jan 2011	10344	6274	1013	346	9233	9604038	10733	31421	194500	320693

12.4.4 Growth in Usage from Spring Semester 2009 through Spring Semester 2011



12.4.5 Internatioinal Usage, January - May 2011

Countries (Top 25) - Full list

Countries		Pages	Hits	Bandwidth
United States	us	1473	17615	867.43 MB
European country	eu	466	4298	126.79 MB
Greece	gr	405	1112	105.10 MB
India	in	346	3097	85.06 MB
Great Britain	gb	311	3943	101.48 MB
Portugal	pt	264	1214	65.39 MB
Australia	au	215	2073	66.10 MB
Canada	ca	148	1747	52.57 MB
Taiwan	tw	128	910	23.99 MB
Japan	jp	102	566	16.13 MB
Brazil	br	94	882	22.68 MB
Turkey	tr	73	932	24.52 MB
Thailand	th	52	470	19.20 MB
France	fr	49	524	16.41 MB
Romania	ro	49	254	6.17 MB
Malaysia	my	48	552	13.72 MB
Ireland	ie	48	356	18.50 MB
Spain	es	43	362	31.28 MB
Indonesia	id	41	521	11.47 MB
Germany	de	38	380	14.11 MB
United Arab Emirates	ae	38	254	10.36 MB
Latvia	lv	36	94	2.15 MB
Philippines	ph	35	520	12.80 MB
Belgium	be	35	255	20.27 MB
Denmark	dk	31	280	15.98 MB
Others		674	8121	243.19 MB

12.4.6 Statistics Revisited

Again, it is important to take the these statistics with a grain of salt. A separate web analytics program available on the server computed more conservative numbers. However, we were unable to gather the output of that program for the desired period of observation. The the usage trend is perhaps the most significant statistic represented. Indeed, the HAS team is still attempting to analyze this trend, hoping to gain more insight into it. The trend is somewhat surprising in that the hypertextbook has not been actively promoted of late, and, as a result of some interruptions to the progress of the project, was not in a condition that the team felt was cohesive enough to market. Thus, any real rise in usage would have been the result of word-of-mouth dissemination.

12.5 Summary

We hope that this short position (*aka*, work in progress) paper will inspire discussion that will help us all better formulate parallel courses forward towards the goal of infusing teaching and learning resources (hypertextbooks) with the visualization tools we all believe enhance student learning.

Bibliography

Christopher M. Boroni, Frances W. Goosey, Michael T. Grinder, and Rockford J. Ross. Engaging Students with Active Learning Resources: Hypertextbooks for the Web. *SIGCSE Bulletin*, 32(1):65–69, March 2001. Paper originally given at SIGCSE 2001.

Stephen Diehl, editor. *Software Visualization: State-of-the-Ary Survey, Lecture Notes in Computer Science 2269*, chapter Hypertextbooks: Animated, Active Learning, Comprehensive Teaching and Learning Resources for the Web. Springer Verlag, 2002. Invited chapter.

Rockford J. Ross. Hypertextbooks and a Hypertextbook Authoring Environment. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, pages 133–137, 30 2008.

Guido Rößling, et al. Merging interactive visualizations with hypertextbooks and course management. *ACM SIGCSE Bulletin*, 38(4), 2006.

Guido Rößling, et al. A visualization-based computer science hypertextbook prototype. *ACM Transactions on Computing Education*, 9(2), June 2009.

List of Authors