Publication P7

Ville Karavirta, Guido Rößling, and Otto Seppälä. 2009. Automatic generation of algorithm visualizations for lectures. Espoo, Finland: Helsinki University of Technology. 11 pages. TKK Research Reports in Computer Science and Engineering B, Report 7 (TKK-CSE-B7). ISBN 978-952-60-3251-1. ISSN 1797-6944.

Automatic Generation of Algorithm Visualizations for Lectures

Ville Karavirta*, Guido Rößling†, and Otto Seppälä‡

October 20, 2009

Abstract

Algorithm animations and visualizations (AVs) aim at making program code or algorithms more understandable by providing a view of the code on a higher level of abstraction. Despite the demonstrated benefits, algorithm visualizations have not been widely adopted in teaching. Presentation tools such as Microsoft PowerPoint and OpenOffice.org Impress are often used by instructors, and algorithm animations are added to the lecture slides. The generation of animations with these tools can be awkward and time-consuming. In this paper, we present a set of tools that allows easy generation of algorithm animations to be used in lecture slides.

Keywords: Algorithm Animation, Animal, Xaal, Lecture Slides, Presentation

1 Introduction

Helping students to understand difficult pieces of code remains a challenge in Computer Science education. By providing a view of the code on a higher level of abstraction, algorithm animations and visualizations (AVs) aim at making the code more understandable. Despite the demonstrated benefits, algorithm visualizations have not been widely adopted in teaching [13]. The integration of visualization in self-study material has been studied in the context of HTML-based hypertextbooks [21]. HTML allows integrating animations in a variety of formats, including Java applets, Flash animations and videos.

One often overlooked way of using algorithm animation is to accompany animations with presentation slides. However, the problem of integrating visualizations in lecture material has not been studied extensively. In a survey study performed by an ITiCSE 2002 Working Group, 79% of the educators listed the "time it takes to adapt visualizations to teaching approach and/or course content" as a substantial impediment for adopting new visualizations to be used on a course [13]. The same difficulties were also found in a recent international survey [10]. Ben-Bassat Levy and Ben-Ari argue that tool developers often do not invest enough research in how pedagogical software can be embedded into a curriculum

*vkaravir@cs.hut.fi, Department of Computer Science and Engineering, Helsinki University of Technology

†roessling@acm.org, CS Department, Technische Universität Darmstadt

‡oseppala@cs.hut.fi, Department of Computer Science and Engineering, Helsinki University of Technology

[1]. While the most commonly used presentation tools (PowerPoint, OpenOffice.org Impress and Keynote) support embedding materials such as video into the lecture slides, the lecturer cannot adapt the animation to the specific lecture case.

For lecture use, the development in AV systems has focused on systems that can be used to give presentations. For example, Alvis [5], ANIMAL [18] and MatrixPro [9] all have features to support their use in lectures (see Section 5). However, in most cases, animations created with AV tools cannot be embedded into the lecture materials since they are implemented as independent applications. Instead, they require the educator to switch between a number of programs during classroom presentation. Some algorithm visualization tools may allow overcoming this limitation by designing the lecture slides inside the algorithm visualization system. But in most cases, this is not desired, feasible or attractive, as the regular presentation tools are far more sophisticated than the AV authoring tools.

In this paper, we present a process to easily generate algorithm animations in lecture slide format with an existing AV system. The paper is organized as follows. In Section 2, we will explain the rationale behind this research. Section 3 introduces the systems used in our approach and Section 4 describes our chosen solution. Section 5 discusses related work, followed by a presentation of the benefits of the solution and outlines areas of future research in Section 6.

2 Motivation

Let us regard a typical scenario in a data structures and algorithms course discussing a new algorithm. In order to use an existing visualization for this algorithm, a suitable presentation must first be found. Here, the teacher can already run into problems. Recent research shows that existing algorithm visualization are often not pedagogically effective and concentrate on simpler algorithms [23]. Even if the teacher finds an existing visualization, there are also other problems that the teacher has to face:

- *Visualizations often concentrate on certain features of the algorithm* and ignore others. The information selected to be shown and the way it is depicted are chosen to emphasize certain points and ignore others. There might be a pedagogical reason for not showing a piece of information, or it might be hidden in order to reduce visual complexity. The skill level and background of the audience also plays a big role and affects what works and what does not.

  As research suggests, the pedagogical style of the visualization might not match the style of the teacher [1]. The final choice of the visualization is likely to be a compromise which contains most of the points the teacher wants to address during the lecture. In some cases, the lecture slides may also have to address limitations of the visualization, as the visualizations themselves can typically not be altered.

- A different problem is that *visualizations often use topic-specific tools*. For a complete course, the teacher might have to use several such visualizations tools. This can result in visualizations that do not share a uniform look and can lead to unnecessary confusion for teachers and students. Additional confusion can be caused by (even subtle) variations in terminology.

- According to results of an international survey, *the classroom set-ups vary a lot*, with the most typical set-up being a class with a computer and a ceiling-mounted projector [13]. Thus, the teacher has to make preparations before the lecture. Applets should be downloaded and local webpages created to account for problems in network connections. In some cases, the visualization software has to be installed, which might even be impossible in some lecture hall set-ups. The safest alternative is often to use a personal laptop for giving the presentations. In contrast, PowerPoint or PDF slides typically work with whatever computer is available.

- Finally, *visualizations often use hard-coded input data*. To illustrate a problem and stimulate self-study, teachers and learners should be able to provide the input data for the algorithm.

Many teachers use generic presentation tools, such as Microsoft PowerPoint, OpenOffice.org Impress or Keynote, for animations of a given algorithm or set of algorithms. These animations are usually created manually for a fixed set of input data. Changing the input presented in one slide may thus require a manual correction of all subsequent slides. Manual generation is very time-consuming and does not scale to other input sets or similar (but slightly different) algorithms, such as a different approach for choosing the pivot element in Quicksort. Additionally, the lack of support for the data structures typically used in computer science makes even the animation of a list- or array-based algorithm very difficult and time-consuming [18].

In a study of the technology and learning expectations of the "net generation", students *"praised PowerPoint's ability to help faculty members convey specific information when used appropriately"* [15]. However, in the same study, students *"expressed significant frustration with faculty members who simply transferred their lecture notes to PowerPoint slides and expected quality learning to occur"*. We expect animations in lecture slides to promote learning through increasing student motivation.

3 Background

This work is a continuation of a previous research prototype [22]. We will provide a brief overview of the previous system and point out its limitations and problems. Next, we will introduce the technologies behind the solution described in the rest of the paper, namely the animation generators and programming API in Animal, and an XML-based algorithm animation language, Xaal.

3.1 First Prototype Implementation

A first prototype implementation of an automatic generation of lecture slides was presented in [22]. The solution, as illustrated in Figure 1, was based on a Java program generating graph descriptions in the GraphViz `dot` format [4]. The graph descriptions were transformed to Scalable Vector Graphics (SVG) using GraphViz. The SVG files were then converted to Open Document Format (ODF) format using XSLT stylesheets. The ODF file could then be opened in OpenOffice.org Impress. More specifically, the Java program generated example cases of the Kruskal algorithm.

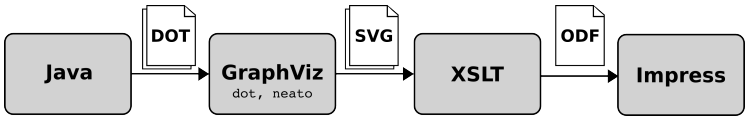

Figure 1: Architecture of the prototype solution

The prototype solution was successful in that it showed this kind of slide generation is possible. It could also automatically generate questions as notes on the slides that a teacher can use to make lectures more interactive. Alas, the approach taken had several problems:

- The use required the installation of third-party tools such as GraphViz and Saxon. This makes it unlikely that teachers adopt it, since cumbersome installation is one of the reasons for not using visualization tools [16].
- In addition to the graph descriptions, the Java program generated other files used in the construction of the ODF slides. This made the XSLT stylesheets quite specific for this use-case; they would not be easy to use in other examples.
- The use of XSLT to generate the slides limited the applicability to simple cases due to the nature and limitations of XSLT.
- Most importantly, the approach supported no reuse of existing animations. As there were a number of animations in repositories for the existing systems, teachers should be able to benefit from those.

3.2 Animal Generators and API

Animal is a system for animating algorithms and data structures. Content can be generated manually using drag and drop in the graphical front-end, using the built-in scripting language AnimalScript [19], and the AlgoAnim Java API. The tool home page provides more than 60 hand-made animations covering most of the basic algorithms and data structures for a typical CS2 course.

Animal also provides a set of animation content generators [17]. Each generator addresses one specific instance of a given algorithm and can be used to generate an animation of that algorithm on-the-fly with only a few mouse clicks. First, the user has to choose an output language, e.g., English, and a programming language, e.g. Java. The former language defines the output language for texts in the animation, while the latter determines the source- or pseudocode shown below the algorithm during the animation.

The user can then choose the concrete type of algorithm, e.g., sorting algorithms, followed by the algorithm, e.g. Bubble Sort. If there are multiple generators for the same algorithm, for example displaying different variants of Bubble Sort, the user has to decide on the best fit. He or she can then configure the generator by providing the input data set. The user can also adjust the visual appearance of the content, for example by changing the color or font settings for the visible elements. After choosing a filename, the animation is created and can be viewed in Animal.

Implementing a new generator is easy to do if one has programming experience in Java. Essentially, the *Generator* interface has to be implemented, which contains mostly methods

that specify metadata, such as the content author, output or programming language. The content generation is triggered by a single method that will typically use the AlgoAnim API [20] for creating the actual animation contents. This is a Java API that is easy to use and supports a large set of primitives and data structures and operations thereon. In addition to primitives and operations, the API also supports the integration of interactive elements, especially questions asked to predict the future behavior or characterize the current state of the algorithm.

The AlgoAnim API was designed to be able to produce output in a variety of formats, although currently only back-ends for the scripting language AnimalScript used by Animal and Xaal (described in the next section) have been implemented.

3.3 Xaal (eXtensible Algorithm Animation Language)

Xaal is an XML language specified for describing algorithm animations. The design of Xaal (eXtensible Algorithm Animation Language) was based on a report of an ITiCSE Working Group [14]. The following will briefly introduce the most important features of Xaal. The reader should note that this text is merely an overview of the language. For a more detailed discussion, see [7, 6], and for the actual XML schemas, see the Xaal website[1].

Graphical Primitives are the basic graphical components that can be composed to represent arbitrarily complex objects, such as a tree data structure. Xaal supports the graphical primitives point, polyline, line, polygon, arc, ellipse, circle and circle segment, square, triangle, rectangle, and text.

Data Structures can be used to specify the visualizations, lowering the effort needed to produce the visualizations. The set of structures in Xaal is: array, graph, list, and tree. To support the different approaches of existing algorithm animation languages, all structures support an optional graphical presentation indicating how the structure should be visualized. This allows even simple viewing software to display complex data structures and correspondingly allows using native visualizations when available.

Animation operations in Xaal have been divided in three groups: graphical primitive transformations (for example, rotate), elementary data structure operations (for example, replace), and abstract data structure operations (for example, insert). Each abstract operation can contain the same operation on a lower level of abstraction as graphical primitive transformations and as elementary data structure operations. However, these are both optional.

In addition to the language specification, Xaal comes with a set of tools that ease enabling the import/export of Xaal in existing algorithm animation systems. These tools allow transforming Xaal animations into other algorithm animation languages. The tools can also be used to add missing information to the animation, for example, instructions on how to render data structures for systems capable of using only graphical primitives.

[1] `http://xaal.org/`

4 Slide Generation

We envisioned a process where the lecture slides could be generated directly from a visualization system. For this, Animal [18] with its collection of animation generators on various topics seemed a suitable choice. The final process is illustrated in Figure 2. The Xaal language implementation in Animal is used to allow generation of Xaal animations using the existing Animal generators. The resulting Xaal document is transformed to lecture slides in ODF format. These slides can then be opened in OpenOffice.org Impress, as illustrated in Figure 3.

Figure 2: Architecture of our solution

4.1 Animal language for Xaal

As described in Section 3, adding a new language to Animal is done by implementing the AlgoAnim API. This was done to a certain extent to have Animal generators output Xaal. The implementation consists of a number of Java classes that generate the required Xaal output for the different primitives and operations of the API, such as graphical primitives and move and scale operations on them.

At its current state, the Animal language for Xaal implements the graphical primitives and operations on them, as well as the generation of source code and interactive elements. Of the data structures in Animal, we have only implemented arrays, and implementation of the other structures remains as a task for the future. However, the implemented features already cover a large portion of the generators available in Animal.

4.2 Transforming Xaal to ODF

In the first prototype, we decided to generate the XML used in ODF documents using a XSL stylesheet. This proved to be more difficult than anticipated. Since we are now transforming Xaal animations and there already is Java support for the language, we decided to generate the ODF contents from Xaal using Java.

For this transformation, we use the ODFDom library, a part of the ODF Toolkit[2]. This library provides a Java API for creating the different elements in the ODF document. Essentially, there is a corresponding Java class for each of the XML elements in an ODF document.

One limitation of our system is the use of discrete steps to show the keyframes in the animation. Although smooth animation is possible in ODF, we did not feel that the benefit of this would be worth the difficulty of implementing it.

Another lack is that we currently lose the information of interactive questions created by the Animal generators. These are questions that, when viewed in Animal, are presented

[2] `http://odftoolkit.org/`

to the student. Our original idea was to include the questions in the lecture notes for the teacher to ask during the lecture, as done in the first prototype. Alas, the version of ODFDom we have used does not support presentation notes and we therefore had to leave this feature open for future development.

4.3 Slides in OpenOffice.org Impress

Our solution provides an effortless way to create customized and customizable lecture animations. They can be customized in the sense that the content generators provided by ANIMAL can be configured in a graphical user interface, allowing the user to adapt the visual appearance and the input data for the algorithm. The lecture animations are also customizable in the sense that the generated slides can be easily modified in OpenOffice.org Impress.

In the resulting slides, it is easy to change things such as the used template or the fonts and colors used for different objects in the animation. Some things are more difficult to change. For example, moving one node in a graph requires moving it in all slides showing that node. However, node styles can be changed so that the node is changed on all slides.

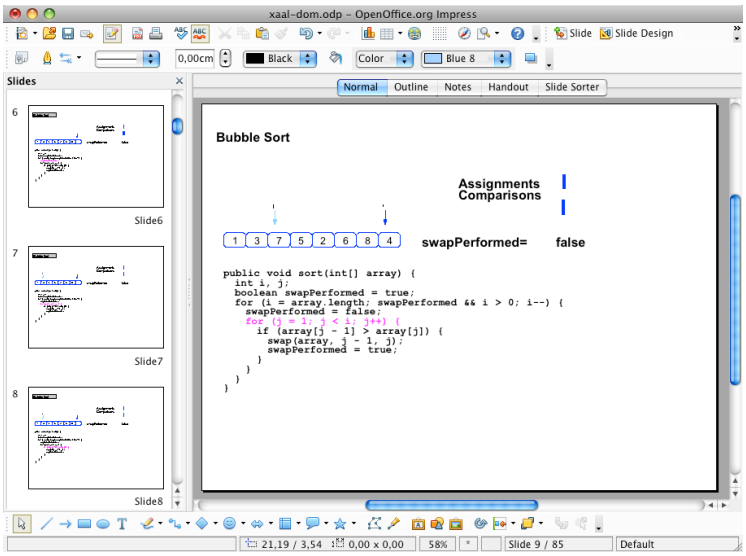

Figure 3: An example of an algorithm animation from the bubble sort generator in ANIMAL transformed to a set of OpenOffice.org Impress slides.

Although we used ANIMAL, one could also use the same Xaal2ODF transformation to export lecture slides from a different tool, for example MatrixPro [9]. Hopefully, this will create a sufficiently easy option for many visualization system developers to include lecture slide generation on the feature list of their system.

5 Related Work

Although the lecture use of algorithm animations has not been studied as intensely as the student use of animations, some relevant related work exists. We introduce the work done on the two approaches taken to integrate algorithm animations to lectures: lecture support offered by AV systems, and slides generated from AV systems. There are also several tools that make it easier to generate lecture slides in OpenOffice.org Impress, PowerPoint or PDF format. However, as they are not concerned with visualizing algorithms, discussing them is beyond the scope of this article.

5.1 Lecture Support in Algorithm Animation Systems

The first approach is to add features to AV systems that support their use in lectures. This approach can be seen especially in ALVIS, Animal, and MatrixPro, all tools that have been used on numerous courses.

ALVIS [5] includes several features that support giving presentations using the tool. First, the tool has a *presentation pointer* which allows pointing to objects in the animation while giving a presentation. Second, the *markup pen* can be used to dynamically annotate the animation. Finally, the presenter can dynamically change the animation as it is executing.

Animal [18] offers a set of features for easy incorporation into the classroom. The generators mentioned before enable animation creation on-the-fly, based on user input. Both the animation speed and the magnification factor can be adapted fluently to the lecture environment. A slider allows fast navigation inside an animation. A table of contents view for animations can be used to quickly jump to points of interest, if these were defined by the animation generator. Finally, Animal was one of the AV pioneers in unbounded bidirectional navigation, enabling lecturers and students to arbitrarily skip ahead or step backwards in the animation.

MatrixPro [9] was originally designed to be used in lectures to demonstrate data structures and algorithms. Features supporting this are the possibility of on-the-fly use, automatic node labeling in data structures, and a library of ready-made data structures such as binary search trees, B-trees, AVL-trees, and red-black-trees.

5.2 Slides from AV systems

We are only aware of one visualization system that is able to export lecture slides. The Leonardo Web builder [3] is a visualization editor that allows generating animation scripts for the Leonardo Web [2] visualization system. Leonardo Web builder can export the animations as PowerPoint slides. In addition, it can export Adobe Flash and animated GIFs. The main difference to our approach is that the created animation scripts can only be used with Leonardo Web, whereas Xaal animations can be used and created with several tools.

6 Discussion

With the approach presented in this article, many of the more than 100 content generators provided in Animal are readily usable to generate lecture slides for the algorithms they

cover. We feel that this, in itself, is a valuable asset for educators. Note that the generators also support custom input and look-and-feel customization by the teacher.

The parts of the process presented can be used together or separately. This means that the Animal generators can be used to output any language supported by the API. Xaal animations generated from Animal can, for example, be integrated into hypertext using the JSXaal viewer [8]. Furthermore, any existing Xaal animation can be transformed to ODF slides. In general, when combining the generation of Xaal from Animal generators with the rest of the Xaal tools, many more options are available:

- **JHAVÉ** The latest version of JHAVÉ [12] includes a Xaal visualizer, enabling it to visualize Xaal animations. Animations in the GaigsXML [11] language used by JHAVÉ can also be transformed into Xaal.
- **JSXaal** The JavaScript Xaal viewer [8] is a JavaScript+HTML implementation of an algorithm animation viewer allowing integration of Xaal animations into HTML.
- **MatrixPro** An extended version of MatrixPro [9] can be used to create animations simply by dragging and dropping keys and nodes into various data structures. It can also export the animation as Xaal, and can import Xaal animations that use data structures.

In Section 2 we discussed problems a teacher faces when finding visualizations for lectures. The approach presented here addresses most of the issues. Neither Animal nor Xaal restrict the use on the scope of data structures and algorithms. Thus, the same approach could be used to generate animations on arbitrary other topics. The ODF slides should open in various classroom set-ups, or PDF version of slides can be generated using OpenOffice.org Impress. Finally, the Animal generators allow changing the input data for the algorithm as well as changing the visual properties of different elements in the visualization. Thus, certain elements can be, for example, hidden by the teacher.

There are still some open issues with our current system. One of them concerns the loss of semantic data in the transformation process. For example, source code lines in Animal are normal text primitives in Xaal (and subsequently in ODF). In addition, all data structures are described using graphical primitives in ODF. Another problem is that the customization of the ODF slides is not very simple for some parts.

In the future, we aim for a more complete Xaal implementation in Animal. Furthermore, the plan is to create more Animal generators, especially for the difficult topics as suggested by Shaffer et al. [23]. Another direction is to create a web application for producing the animations where a teacher could provide input for the Animal generators and get the output in a user-preferred format all within a browser. This would hopefully make the generation of animations easy enough for teachers to use, and it could be used by students as well. It should be noted that the same approach could be taken to generate slides for Microsoft PowerPoint using, for example, the OpenXML4J library[3].

Finally, we encourage visualizers to create their animations as Animal generators to allow for the reusability of the animations. Furthermore, AV system developers are urged to add Xaal support for their AV system.

[3] `http://openxml4j.org/`

References

[1] R. Ben-Bassat Levy and M. 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.

[2] V. Bonifaci, C. Demetrescu, I. Finocchi, G. F. Italiano, and L. Laura. Portraying algorithms with Leonardo Web. In M. Dean, Y. Guo, W. Jun, R. Kaschek, S. Krishnaswamy, Z. Pan, and Q. Z. Sheng, editors, *Web Information Systems Engineering / WISE 2005 Workshops*, volume 3807 of *Lecture Notes in Computer Science*, pages 73–83. Springer, 2005.

[3] V. Bonifaci, C. Demetrescu, I. Finocchi, and L. Laura. Visual editing of animated algorithms: the Leonardo Web builder. In *AVI '06: Proceedings of the working conference on Advanced Visual Interfaces*, pages 476–479, New York, NY, USA, 2006. ACM.

[4] J. Ellson, E. Gansner, L. Koutsofios, N. S. C., and G. Woodhull. Graphviz open source graph drawing tools. *Lecture Notes in Computer Science*, 2265/2002:594–597, 2002.

[5] C. D. Hundhausen and S. A. Douglas. Low-fidelity algorithm visualization. *Journal of Visual Languages and Computing*, 13(5):449–470, Oct. 2002.

[6] V. Karavirta. XAAL – extensible algorithm animation language. Master's thesis, Department of Computer Science and Engineering, Helsinki University of Technology, December 2005. Available online at `http://www.cs.hut.fi/Research/SVG/publications/karavirta-masters.pdf`.

[7] V. 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, 2007. Elsevier Science Publishers B. V.

[8] V. Karavirta. Seamless merging of hypertext and algorithm animation. *ACM Transactions on Computing Education (TOCE)*, 9(2):1–18, 2009.

[9] V. Karavirta, A. Korhonen, L. Malmi, and K. Stålnacke. MatrixPro – A tool for on-the-fly demonstration of data structures and algorithms. In *Proceedings of the Third Program Visualization Workshop*, pages 26–33, The University of Warwick, UK, July 2004. Department of Computer Science, University of Warwick, UK.

[10] E. Lahtinen, H.-M. Järvinen, and S. Melakoski-Vistbacka. Targeting program visualizations. In *ITiCSE '07: Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 256–260, New York, NY, USA, 2007. ACM.

[11] T. Naps, M. McNally, and S. Grissom. Realizing XML-driven algorithm visualization. In *Proceedings of the Fourth Program Visualization Workshop (PVW 2006)*, volume 178 of *Electronic Notes in Theoretical Computer Science*, pages 129–135, Amsterdam, The Netherlands, 2007. Elsevier Science Publishers B. V.

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

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

[14] T. L. Naps, G. Rößling, P. Brusilovsky, J. English, D. Jarc, V. Karavirta, C. Leska, M. McNally, A. Moreno, R. J. Ross, and J. Urquiza-Fuentes. Development of XML-based tools to support user interaction with algorithm visualization. *SIGCSE Bulletin*, 37(4):123–138, December 2005.

[15] G. R. Roberts. Technology and learning expectations of the net generation. In D. G. Oblinger and J. L. Oblinger, editors, *Educating the Net Generation*, chapter 3, pages 3.1–3.7. Educause, 2005.

[16] R. J. Ross and M. T. Grinder. Hypertextbooks: Animated, active learning, comprehensive teaching and learning resource for the web. In S. Diehl, editor, *Software Visualization: International Seminar*, pages 269–283, Dagstuhl, Germany, 2002. Springer.

[17] G. Rößling and T. Ackermann. A Framework for Generating AV Content on-the-fly. In *Proceedings of the Fourth Program Visualization Workshop (PVW 2006)*, volume 178, pages 23–31, Amsterdam, The Netherlands, 2007. Elsevier Science Publishers B. V.

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

[19] G. Rößling, F. Gliesche, T. Jajeh, and T. Widjaja. Enhanced Expressiveness in Scripting Using AnimalScript 2. In *Proceedings of the Third Program Visualization Workshop*, pages 10–17, The University of Warwick, UK, July 2004.

[20] G. Rößling, S. Mehlhase, and J. Pfau. A Java API for Creating (not only) AnimalScript. In *Proceedings of the Fourth Program Visualization Workshop (PVW 2006)*, volume 224, pages 15 – 25, Amsterdam, The Netherlands, 2009. Elsevier Science Publishers B. V.

[21] G. Rößling, T. Naps, M. S. Hall, V. Karavirta, A. Kerren, C. Leska, A. Moreno, R. Oechsle, S. H. Rodger, J. Urquiza-Fuentes, and J. A. Velázquez-Iturbide. Merging interactive visualizations with hypertextbooks and course management. *SIGCSE Bulletin*, 38(4):166–181, 2006.

[22] O. Seppälä and V. Karavirta. Work in progress: Automatic generation of algorithm animations for lecture slides. In *Proceedings of the Fifth Program Visualization Workshop (PVW 2008)*, volume 224, pages 97–103, Amsterdam, The Netherlands, 2009. Elsevier Science Publishers B. V.

[23] C. A. Shaffer, M. Cooper, and S. H. Edwards. Algorithm visualization: a report on the state of the field. In *SIGCSE '07: Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, pages 150–154, New York, NY, USA, 2007. ACM Press.