Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 224 (2009) 105–114

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Towards Seamless Merging of Hypertext and Algorithm Animation

Ville Karavirta[1]

Department of Computer Science and Engineering
Helsinki University of Technology
Espoo, Finland

Abstract

The integration of algorithm animations into hypertext is seen as an important topic today. This paper will present a prototype algorithm animation viewer implemented purely using HTML and JavaScript. The viewer is capable of viewing animations in Xaal (eXtensible Algorithm Animation Language). This solution is extremely suitable to be used in hypertext learning material.

Keywords: Algorithm animation, online learning, merging of hypertext and algorithm animation, Xaal

1 Introduction

Online learning material that students use by themselves is one of the typical usages of algorithm animation (AA). Thus, the integration of algorithm animations into hypertext is seen as an important topic today. The hope is that by making it easier to use AA in hypertext, algorithm animations will become more widely adopted in teaching. This could then solve the problem AA has been fighting for years: teachers believe that AA is beneficial but they do not use it in their teaching [9]. In 2006, a working group titled *Merging interactive visualizations with hypertextbooks and course management* convened at ITiCSE 2006 to consider how algorithm visualizations could and should be merged with hypertext. In the working group report, the features seen important included seamless visualization integration, increasing student engagement, providing a richer learning environment, integration of CMS features, and aesthetics. [15]

The technologies for building interactive web applications have evolved fast in the past few years. This has made it possible to implement complex applications

[1] Email: `vkaravir@cs.hut.fi`

doi:10.1016/j.entcs.2008.12.054

online. As a result, we have seen a surge of many typical desktop applications being implemented as rich internet applications (or, RIAs). These applications include mail clients, office software, and even photo-editing software. The main benefits of online applications are the ease of which they can be taken into use and the ease of maintenance from the developers perspective.

When the goal is to seamlessly merge algorithm visualizations with hypertext, a natural future direction for algorithm animation systems is to implement them as RIAs. This paper introduces one such solution. We will present a prototype algorithm animation viewer implemented purely using HTML and JavaScript. The viewer is capable of viewing animations in XAAL (eXtensible Algorithm Animation Language), a language designed to allow easy transformation of AAs between various formats [3]. This solution is extremely suitable to be used in hypertext learning material. In the end, our goal is to create an interactive system integrated with hypertext that supports several current algorithm animation description languages.

The paper is organized as follows. First, Section 2 introduces related research. Section 3 discusses the requirements for algorithm animation systems. Section 4 in turn describes the implementation of the new AA viewer. Finally, Section 5 discusses the suitability of this new approach and Section 6 provides conclusions and some possible future directions.

2 Related Work

Algorithm animation has been an active area of research. From the perspective of this paper, the most important research directions are merging AA and hypertext and developing a common, XML-based algorithm animation language.

Early work on algorithm visualization in hypertextbooks has been done by Ross [13]. In Ross's hypertextbooks, the inclusion of visualizations is done using Java applets. This is currently a common way also used in, for example, TRAKLA2 [4], ViLLE [12], WinHIPE [11], and a number of topic-specific visualizations. Another popular method at the moment is Java WebStart where users launch Java applications from the web. This approach is used, for example, in ANIMAL [14], Jeliot [5], and JHAVÉ [7].

In ITiCSE 2006, a working group considered how algorithm visualizations should be merged with hypertext. The group considered visualization based hypertextbooks an important factor in promoting algorithm animation adoption in teaching. In the working group report, the features of such hypertextbooks the working group considered important were seamless visualization integration, increasing student engagement, providing a richer learning environment, integration of CMS features, and aesthetics. [15]

Another related research topic is the integration of algorithm animation systems. One possible way to achieve this integration has been taken in the JHAVÉ algorithm visualization environment. JHAVÉ allows AA system developers to add their systems as visualization plugins into JHAVÉ. This way a common user interface for the end-user (typically, a student) is achieved. Another approach to integration, focusing on the teacher's point of view, is developing a common format for the algorithm animation systems. In AV scope, this problem has been discussed in another ITiCSE working group. The group's report gives examples of a common AA language and suggestions on an architecture to implement it [10].

One implementation of the ideas of the group is XAAL (eXtensible Algorithm Animation Language) [3]. XAAL supports describing animations on different levels of abstraction: using graphical primitives and transformations on them, or using data structures and operations on them. The goal of XAAL and the tools supporting it has been to allow easy transformation of AAs between various formats/systems. The import and export features of visualization systems is a significant research problem even in the wider scope of Software Visualization [1].

From the purely technical point of view, several rich internet application (or, RIA) technologies have been introduced lately. These technologies allow creating complex applications that run in web browsers. In this work, we will focus on JavaScript. However, we will introduce some alternatives in Section 4. On the field of JavaScript, a multitude of libraries aiding in web development have been developed, and new ones are popping up constantly. Some of the most well-known libraries include Dojo, mootools, Prototype, Scriptaculous, jQuery, and Google Web Toolkit, just to mention a few.

When discussing algorithm animation in the context of education [2], one cannot ignore the importance of user engagement. It has been shown that when users interact with algorithm animations, it has a positive impact on their learning [2]. The levels of engagement were specified by Naps et al. by introducing a taxonomy of engagement [8]. The levels are *viewing*, *responding*, *changing*, *constructing*, and *presenting*. *Viewing* is passive watching of an animation where a student only controls the visualization execution. In *responding*, the student is engaged by asking questions about the visualization. *Changing* requires the student to modify the visualization, for example, by changing the input data. In *constructing*, the student is required to construct his/her own algorithm animation. At the highest level, *presenting*, the student presents a visualization for an audience.

3 Requirements for an Algorithm Animation System

Over the years, a lot of research on the requirements of algorithm animation systems has been carried out. Rößling and Naps introduced pedagogical requirements for algorithm visualizations (AV) [16]. In another article, they provide more guidelines for AV systems [17]. The following summarizes the requirements of these two articles. We have numbered the requirements to help referring to them later in the article.

- The system's platform should be chosen to allow the widest possible target audience. [R1]
- The system should support visualization rewinding so that users can return to

[2] In fact, the most common application area for algorithm animation lies in context of education [1].

the place where they lost track of the content. [R2]

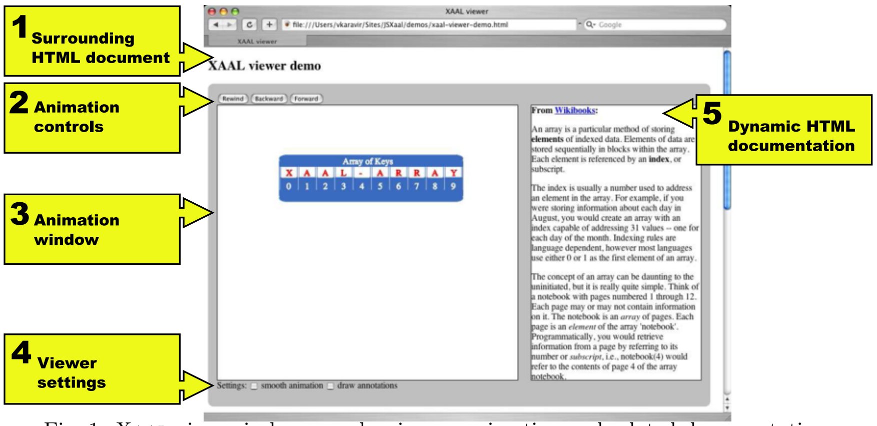
- Learners should be able to adapt the display to their current environment. This includes the choice of display background color to account for diverse lighting situations, transition speed and display magnification. [R3]
- The AA system should preferably be a general-purpose system instead of a topic-specific system due to the chance for reuse and better integration into a given course. The main benefit of general-purpose systems is the ability to offer a common interface to a large number of animations. [R4]
- The user should be offered the choice between smooth visualization transitions and discrete steps. A break between consecutive steps, or at least a pause button, should also be provided. [R5]
- The visualizations should include documentation that accompany the visualization. There are several types of documentation: static documentation, dynamic documentation aware of the algorithm's state, and pedagogically dynamic documentation that is aware of the algorithm's state as well as the expertise of the student. [R6]
- Asking questions about the algorithms behavior in following states should be supported. The questions should incorporate feedback. [R7]
- The system should be integrated with a database for course management facilities. The database can be used, for example, to store the points received by answering questions. [R8]
- The system should allow users to provide custom input to the algorithm. [R9]
- The visualization should offer a structural view of the algorithm's main parts that can also be used to jump to associated visualization steps. [R10]

The articles also include two more requirements. One system requirement is that for the visualization author, the system should make it possible to group questions based on the question topic area and to assign a certain number of points to each question and inform the learners on their progress. Another requirement is to include reusable visualization modules, thus aiding in the authoring of AV. Since both of these are requirements for the author, we do not consider it relevant in this case when building an algorithm animation viewer.

4 Implementation

Based on the requirements for an algorithm animation viewer, we implemented a prototype of such a system using only HTML and JavaScript. As the whole viewer will be running inside the user's browser without any additional plugins, every computer equipped with a modern browser can use the viewer (requirement R1). The implemented prototype can be easily incorporated into web material using Xaal animations as the source data. The following describes the implemented features and, for the interested reader, the technologies used.

Figure 1 shows the animation viewer in the Safari browser. In the figure, arrow 1


1 Surrounding HTML document
2 Animation controls
3 Animation window
4 Viewer settings
5 Dynamic HTML documentation
XAAL viewer
file:///Users/vkaravir/Sites/JSXaal/demos/xaal-viewer-demo.html
Google
XAAL viewer
Rewind
Backward
Forward
Array of Keys
X A A L - A R R A Y
0 1 2 3 4 5 6 7 8 9
Settings: smooth animation draw annotations

XAAL viewer demo

From Wikibooks:

An array is a particular method of storing **elements** of indexed data. Elements of data are stored sequentially in blocks within the array. Each element is referenced by an **index**, or subscript.

The index is usually a number used to address an element in the array. For example, if you were storing information about each day in August, you would create an array with an index capable of addressing 31 values -- one for each day of the month. Indexing rules are language dependent, however most languages use either 0 or 1 as the first element of an array.

The concept of an array can be daunting to the uninitiated, but it is really quite simple. Think of a notebook with pages numbered 1 through 12. Each page may or may not contain information on it. The notebook is an *array* of pages. Each page is an *element* of the array 'notebook'. Programmatically, you would retrieve information from a page by referring to its number or *subscript*, i.e., notebook(4) would refer to the contents of page 4 of the array notebook.

Fig. 1. XAAL viewer in browser showing an animation and related documentation.

points to the surrounding HTML document. This document can contain any HTML. Arrow 2 points to the animation controls. Here, we have the controls to rewind and move backwards and forwards in the animation (fulfilling requirement R2). Arrow 3 points to the actual animation window where the contents of the animation are visualized. Arrow 4, in turn, indicates the settings panel for the animation viewer. Finally, arrow 5 points to the HTML documentation that is included in the XAAL document and shown next to the visualization. The appearance of the viewer can be easily changed using a different CSS stylesheet. Through CSS, one can also change the positions and sizes of the various parts of the viewer (partially fulfilling R3). To completely fulfill the requirement, features like speed and magnification changing should also be supported.

Listing 1 gives an example on how to add a XAAL animation into an HTML document. As can be seen, it is quite simple and requires only a few lines of code. Multiple animations can be embedded on the same page by repeating lines 1 and 5 of Listing 1 with different data.

```
1 <div id="animation" class="jsxaal"></div>
2 ...
3 <script type="text/javascript">
4   Event.observe(window, 'load', function() {
5     new JSXaalViewer("slides.xml", "animation", { showNarrative: true });
6   });
7 </script>
```

Listing 1: Example of including a XAAL animation in hypertext.

In addition to the code in Listing 1, the JavaScript files for the viewer need to be loaded, which adds another couple of lines. Listing 2 shows these requirements. The first five lines of these are libraries used by the viewer (see Section 4.1 for details about the libraries), while the last line is the actual XAAL viewer.

```
1 <script src="lib/prototype/prototype.js" type="text/javascript"></script>
2 <script src="lib/pgf/pgf-core-min.js" type="text/javascript"></script>
3 <script src="lib/pgf/pgf-renderer-min.js" type="text/javascript"></script>
4 <script src="lib/scriptaculous/scriptaculous.js" type="text/javascript"></script>
5 <script src="lib/scriptaculous/effects.js" type="text/javascript"></script>
6 <script src="dist/jsxaal-core-min.js" type="text/javascript"></script>
```

Listing 2: Example of loading the libraries required to add a XAAL animation in

hypertext.

The graphical primitives of XAAL make it possible to use the viewer not only for animation of data structures and algorithms, but everything that can be visualized using graphical primitives. Thus, requirement R4 is met.

As mentioned in the previous section, smooth animation should be optional for the end-user. Thus, our viewer includes the option for smooth animation to be toggled on or off. This can be done using one of the viewer settings (arrow 4 in Figure 1). This was requirement R5.

Since the whole animation viewer is based on HTML, integration with hypertext is simple and natural. Static documentation can be provided outside the viewer. As Listing 1 showed, including XAAL animations in hypertext requires only a couple of lines of HTML and JavaScript. There is also another method of including hypertext when using XAAL animations; each step in the animation is allowed to include a description that can be arbitrary XHTML. Documentation added this way is shown in area labeled 5 in Figure 1. This documentation is dynamic in nature, as it is different for every state of the animation. The only unsupported type of documentation mentioned in the requirements (R6) is dynamic documentation based on user's experience.

Requiring users to respond to questions during the animation was another requirement for an AA viewer (R7). Currently, the viewer supports some typical question types such as true/false questions and multiple choice questions. Since XAAL has no way to specify questions, we have implemented support for the question specification of the GaigsXML algorithm animation language [6]. To connect to a database to submit the students' responses, AJAX (or, Asynchronous JavaScript And XML) calls can be made. Thus, requirement R8 is easy to implement on the client-side. However, it requires some interface on the server side which we haven't considered in this paper.

Allowing users to specify their own input was requirement R9. Again, since we are working with HTML and JavaScript, allowing users to specify their own input data for the viewer is extremely simple in cases where the animation is using data structures. This is because any scripts included in the HTML document can interact with the animation viewer. For example, Listing 3 gives an example how to add a textfield to an HTML page that adds the user-given data into the binary search tree in the animation.

```
<input id="insValue" type="text"></input>
  <button id="insButton">Insert</button>
...
<script type="text/javascript">
  Event.observe($('insButton'), 'click', function() {
    var bst = viewer.dsStore.get("bst");
    bst.insert($('insValue').value);
    bst.draw(viewer.renderer);
</script>
```

Listing 3: Example of allowing user input for algorithms.

4.1 Underlying Technologies

The implementation of the viewer is based on three JavaScript libraries. The lowest level of these libraries is Prototype[3], which offers, for example, Ajax support as well as advanced features for dynamically manipulating the client-side HTML. The visualizations are drawn using Prototype Graphic Framework (PGF)[4], a Prototype-based framework that allows drawing arbitrary data on various browsers. PGF supports multiple rendering technologies for different browsers: Scalable Vector Graphics (SVG), HTML Canvas element, and Vector Markup Language (VML). These different renderers can be used through one programming interface. The animation features in our viewer use Scriptaculous[5], a Prototype-based animation framework. The animation is achieved by extending Scriptaculous's effects to modify graphical objects drawn using PGF.

When discussing web applications, the size of the files and the loading time are essential. Table 1 shows the load times and file sizes of the different components used to implement the XAAL viewer. The total size is slightly over 400 kilobytes. This size can be reduced by minimizing and compressing the files.

Table 1
Load times and file sizes of the different components needed for the viewer.

Component	**Load time (in ms)**	**Size (in kb)**
Prototype	107	124.1
Scriptaculous	177	124.7
Prototype Graphic Framework	72	88.7
XAAL viewer	48	64.7
Total	404	402.2

5 Discussion

The viewer supports the XAAL specification only partially. At the moment of writing, all the graphical primitives and data structures are supported. However, not all animation operations of the specification are supported.

Some of the requirements are not implemented at this point. Some user interface components should be added. These include the options to change the speed and magnification of the visualization (R3) as well as visualizing the algorithm's structure (R10). All these could be implemented with reasonable effort.

Connecting to a database to course management facilities (R8) can be done using AJAX calls. This, however, requires server-side support as well. An interesting new technology that could make implementing this quite simple, is Aptana

[3] `http://www.prototypejs.org`
[4] `http://prototype-graphic.xilinus.com/`
[5] `http://script.aculo.us`

Jaxer[6]. Jaxer is an AJAX server that allows running the same JavaScript code and manipulating the same DOM both on server-side and client-side.

There are still some problems unsolved. First, platform specific problems do arise, although the JavaScript libraries make writing browser independent code a lot easier. Currently, the implementation has been used in Firefox, Safari, Opera, and SeaMonkey but does not work in Internet Explorer. However, there should not be any major impediments in fixing this in IE. In addition, using JavaScript libraries other than Prototype in the HTML document can cause problems. However, this is not usual in current learning environments. Another problem is that due to the nature of JavaScript, all the source code is available to the student. Thus, any client-side assessment results cannot be trusted if such a system is to be used in evaluating students.

5.1 Alternative RIA Technologies

When building rich internet applications, JavaScript is not the only choice. In fact, there is an increasing number of promising technologies available. The discussion of all of these is not possible in the scope of this paper. However, the following mentions some of the most potential candidates.

Adobe's Flash and Flex provide technology for building cross-platform RIAs. The tools for developing applications are quite sophisticated and powerful. However, the tools are commercial software products developed by Adobe. Another rising technology is Microsoft Silverlight, which uses a lot of the same technologies as the .NET framework making it suitable for developers familiar with .NET. However, Silverlight is not cross-platform compatible. Finally, we mention JavaFX, a family of products from Sun Microsystems based on Java technology. However, this technology is not ready for production use at the moment. On a positive side, Sun plans on releasing parts of the JavaFX family as open source.

So, why did we choose the JavaScript road? First, by using JavasScript we do not depend on software provided by any corporation but are using open source libraries. Second, JavaScript works on all platforms without any plugins, whereas, for example, Silverlight is not available on Linux at the time of writing. In addition, our approach can use any server side components. Finally and most importantly, for the JavaScript approach, the technology is mature, widely used, and supported by an ever-growing number of useful libraries.

6 Conclusions and Future Research

In this article, we have introduced a solution for using algorithm animations in hypertext online material. Our solution is a pure HTML and JavaScript implementation of an algorithm animation viewer that fulfills most of the requirements for an AV system. In the future, we hope to be able to use this system in actual material used by learners. At this point, we are not aware of any similar systems being

[6] http://www.aptana.com/jaxer

implemented and we see this as an important step towards the seamless merging of AV and hypertext called for by the ITiCSE 2006 working group.

There are naturally many more possible features that could be implemented. Implementing the rest of the requirements and to support the complete XAAL specification are high on our wish list. However, the nature of HTML and JavaScript offers some unusual possibilities. The following is a list of the most interesting future development ideas.

- In the current version, there is already support for drawing annotations on the animation. In the future, we could store these annotations and then later show them for the same student, or even share the annotations between students.
- We could support opening animations in different formats directly in the browser.
- Due to the nature of JavaScript, replacing functions on the fly is trivial. This would allow creation of automatically assessed exercises where the student is required to code some algorithm using the data structures in the viewed animation.
- The current solution requires internet access if used in evaluation. However, with cutting edge technologies like Google Gears or Dojo.Offline, offline usage of animations where assessment/results are submitted when the student is online, could be developed.

References

[1] Diehl, S., "Software visualization: Visualizing the Structure, Behaviour, and Evolution of Software," Springer New York, 2007.

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

[3] Karavirta, V., "Facilitating Algorithm Animation Creation and Adoption in Education," Licentiate's thesis, Helsinki University of Technology (2007), available online at http://www.cs.hut.fi/Research/SVG/publications/karavirta-lis.pdf.

[4] Malmi, L., V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä and P. Silvasti, *Visual algorithm simulation exercise system with automatic assessment: TRAKLA2*, Informatics in Education **3** (2004), pp. 267–288.

[5] Moreno, A., N. Myller, E. Sutinen and M. Ben-Ari, *Visualizing programs with Jeliot 3*, in: *Proceedings of the International Working Conference on Advanced Visual Interfaces* (2004), pp. 373 – 376.

[6] Naps, T., M. McNally and S. Grissom, *Realizing XML-driven algorithm visualization*, in: *Proceedings of the Fourth Program Visualization Workshop (PVW 2006)*, Electronic Notes in Theoretical Computer Science **178** (2007), pp. 129–135.

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

[8] Naps, T. L., 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** (2003), pp. 131–152.

[9] Naps, T. L., G. Rößling, J. Anderson, S. Cooper, W. Dann, R. Fleischer, B. Koldehofe, A. Korhonen, M. Kuittinen, C. Leska, L. Malmi, M. McNally, J. Rantakokko and R. J. Ross, *Evaluating the educational impact of visualization*, SIGCSE Bulletin **35** (2003), pp. 124–136.

[10] Naps, T. L., 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** (2005), pp. 123–138.

[11] Pareja-Flores, C., J. Urquiza-Fuentes and J. Ángel Velázquez-Iturbide, *WinHIPE: an ide for functional programming based on rewriting and visualization*, ACM SIGPLAN Notices **42** (2007), pp. 14–23.

[12] Rajala, T., M.-J. Laakso, E. Kaila and T. Salakoski, *Effectiveness of program visualization: A case study with the ville tool*, Journal of Information Technology Education: Innovations in Practice **7** (2008), pp. 15–32.

[13] Ross, R. J. and M. T. Grinder, *Hypertextbooks: Animated, active learning, comprehensive teaching and learning resource for the web*, in: S. Diehl, editor, *Software Visualization: International Seminar* (2002), pp. 269–283.

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

[15] Rößling, G., 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** (2006), pp. 166–181.
URL http://doi.acm.org/10.1145/1189136.1189184

[16] Rößling, G. and T. L. Naps, *A testbed for pedagogical requirements in algorithm visualizations*, in: *Proceedings of the 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'02* (2002), pp. 96–100.

[17] Rößling, G. and T. L. Naps, *Towards intelligent tutoring in algorithm visualization*, in: *Second International Program Visualization Workshop* (2002), pp. 125–130.