



ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in
Theoretical Computer
Science

Electronic Notes in Theoretical Computer Science 178 (2007) 79–87

www.elsevier.com/locate/entcs

Integrating Algorithm Visualization Systems

Ville Karavirta¹

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

Abstract

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 Visualization (AV) aims at making the code more understandable. However, teachers consider producing AVs with the existing tools to require too much time and effort to be worthwhile.

One way to lower this effort is to allow data exchange between AV systems. This paper continues the work of the ITiCSE Working Group (WG) “Development of XML-based Tools to Support User Interaction with Algorithm Visualization”. The WG aimed at specifying a common language for AV systems. We analyzed a number of existing AV languages and came up with requirements for a common language. Based on these requirements and the previous work by the WG, this paper defines a new AV language. Furthermore, this study describes a set of tools that allow data exchange between some of the existing AV systems. This data exchange gives teachers more choices and ready-made examples to be used in teaching.

Keywords: algorithm animation, AV system integration, xml, data exchange

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 Visualization (AV) aims at making the code more understandable. However, there is still speculation about its effectiveness in learning [3]. Recent studies indicate that to be *educationally effective* (*i.e.* aid students’ learning) algorithm visualizations need to be more than passive animations; they must require users to interact with the animation [3]. This interaction can, for example, require the user to respond to multiple-choice questions during the animation or to construct an algorithm animation by using a visualization.

However, algorithm visualizations have not been widely adopted in teaching. One of the main reasons is the time and effort required to find and adopt such a

¹ Email: vkaravir@cs.hut.fi

system (or ready-made examples) and to design, create, and integrate the visualizations [8]. Thus, teachers need to have easier ways to find and produce visualizations that provide the needed interaction. For meeting these needs, several AV systems have been developed. Some of the systems that are currently being developed are ANIMAL [10], JAWAA [1], and MatrixPro [6]. These systems provide different ways to create the animations as well as different types of interaction. Thereby, integrating these systems could be beneficial, for example, by allowing the teacher to select the system based on the level of interaction it provides.

One way to achieve this integration is to define a common language for the algorithm visualization systems. This was the topic of one of the working groups at the Conference on Innovation and Technology in Computer Science Education (ITiCSE) 2005. The working group aimed at defining XML specifications for the various aspects of AV. The working group's report [9] provides examples of these specifications as well as guidance on how to use the specifications in existing visualization systems. This paper refers to this group and its work as ITiCSE XMLWG or as working group.

The working group did a lot of good work in coming up with XML specifications for different aspects of AV. However, a lot of the work remains on the level of examples instead of concrete specifications. One idea of the working group was to continue the work in the future. The work presented in this paper builds on the specifications and examples provided by the working group and introduces more concrete specifications in the form of a language, XAAL (eXtensible Algorithm Animation Language), defined to be used in data exchange between algorithm animation systems. In addition, this paper introduces a set of tools that will hopefully benefit AV system developers. The aim of the tools is to allow data exchange between the current AV systems as well as support other useful export formats from the systems.

In the following, Section 2 briefly introduces the main features of the language. For a more specific documentation, see the XAAL website at <http://www.cs.hut.fi/Research/SVG/XAAL/>. Section 3 in turn describes the tools that support the data exchange. Finally, Section 4 discusses the usefulness of such a language and tools as well as looks into the future.

2 eXtensible Algorithm Animation Language

XAAL (eXtensible Algorithm Animation Language) is defined as an XML (Extensible Markup Language) language by specifying the allowed document structure. XML makes it easy for software to process data using the multitude of different tools and architectures available today. In addition, transforming XML documents to different XML formats or text is relatively simple and flexible using XSLT (Extensible Stylesheet Language Transformations).

An important aspect of defining the language has been the need of transformation between different existing algorithm animation languages. To find out the requirements for a language used in data exchange, a survey of the existing de-

scription languages was made [4]. Based on this survey, a taxonomy of algorithm animation languages was defined [5]. This taxonomy indicates that the features of an algorithm animation language can be roughly divided into three categories: data structures, graphical primitives, and animation. Problems arise when trying to exchange data between tools that have different approaches to AV. For example, ANIMAL [10] describes animations using mostly graphical primitives, whereas MatrixPro [6] uses only data structures. These different approaches have been taken into account when defining XAAL.

In this section, we 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 [4] and for the actual XML schemas, see the XAAL website.

2.1 Graphical Primitives

The basic graphical components that can be composed to represent arbitrarily complex objects (*e.g.*, a tree data structure) are graphical primitives. The graphical primitives in XAAL are as specified by the ITiCSE XMLWG [9], where the following have been defined: point, polyline, line, polygon, arc, ellipse, circle and circle-segment, square, triangle, rectangle, and text.

Other features specified by the working group are the definition of reusable *shapes* from the graphical primitives and changing the visual appearance of the graphical primitives using *reusable styles*. Both of these are intended to aid the creation of more complex primitives. Listing 1 in Appendix A gives an example of a shape definition that uses graphical primitives.

2.2 Data Structures

XAAL supports the usage of data structures to specify the visualizations, lowering the effort needed to produce them. The set of structures is basically the same as, for example, in JAWAA [1]: array, graph, list, and tree. The content of these data structures is described using nodes (or indices in case of an array) and edges connecting the nodes. The structures can form arbitrarily complex hierarchies. On the other hand, the simplest kind of structure can be a string or a number. Moreover, to support the different approaches of existing algorithm animation languages, all structures support an optional graphical presentation indicating how the structure should be visualized. Listing 2 in Appendix A shows an example of an array definition.

2.3 Animation

A crucial part of the algorithm animation language is the animation functionality. In the following, we will introduce the elements available in XAAL for defining the animations. The *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).

The operations for manipulating graphical primitives use the format specified by the ITiCSE XMLWG [9]. These include operations such as show, hide, move, rotate, and scale. Listing 3 in Appendix A is an example of a rotate operation.

The *elementary data structure operations* available in XAAL are create, remove, replace, and swap. To be consistent with the structure definitions, every operation can have an optional part describing how the operation should be visualized using graphical primitive animation.

Abstract data structure operations are operations that depend on the semantics of the target structure. In XAAL, the available operations are insert, delete, and search. As with the elementary data structure operations, these operations can have an optional part describing how the operation should be visualized using graphical primitive animation. In addition, for systems that do not know the semantics of the abstract operation, the behavior can be optionally described using elementary data structure operations. Listing 4 in Appendix A shows an example of a delete operation.

2.4 Schema Specification

We have defined an XML Schema for XAAL. To make the language more modular, we have divided the schema into several XML Schema documents. This kind of modularity makes it possible to more easily change or reuse some parts of this language in other languages.

3 Implementation

Our objective was to implement XAAL in a modular way that could be useful for other AV system developers in their aims at implementing the ITiCSE XMLWG specifications. In this section, we will introduce two different processing pipelines to add XAAL support into existing algorithm animation systems. In addition, we will briefly describe a prototype implementation of XAAL and several transformations.

3.1 Object hierarchy

The first processing solution is an architecture discussed by the ITiCSE XMLWG to implement its specifications. This architecture is represented in Figure 1. The basic idea is to have one XAAL *parser* that can be used by multiple algorithm animation systems. This parser generates a Java *object hierarchy*. In addition, there is a part of software that can *serialize* the object hierarchy to a XAAL document.

The existing AV systems can then implement *adapters* that convert the XAAL object hierarchy into an animation in that particular system. By implementing a *generator*, the existing systems can generate the object hierarchy and serialize it as XAAL.

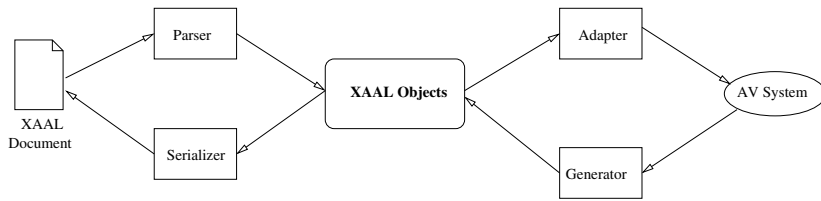


Fig. 1. Integrating XAAL with existing AA systems using an object hierarchy.

This solution requires no major modifications to the existing systems, and thus the workload of implementing XAAL remains fairly low. Another advantage is that the document has to be parsed only once. There is, however, one extra step in the process compared to the direct approach of parsing the XAAL document directly into the AV system. However, implementing a XAAL parser for each system would not be sensible, and thus the extra processing is not considered a major issue.

3.2 XSLT Processing

Another way to integrate XAAL with existing systems is to transform it to a format of the target system using XSLT (represented in Figure 2). This method provides a simple solution to import XAAL documents into existing AV systems that have an algorithm animation language. It can also be used to export different formats from a system that supports XAAL.

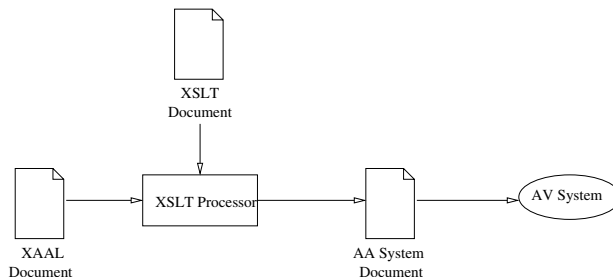


Fig. 2. Integrating XAAL with existing AA systems using XSL stylesheet processing.

The benefit of this approach is that the XSLT stylesheets are quite simple to write for a person who knows the syntax of XAAL, the target language, and XSLT. Moreover, the target system need not be changed at all. This makes it possible to integrate XAAL with systems that are not open-source. On the negative side, this approach requires parsing of three files: the stylesheet, the XAAL document, and the generated AV system document.

3.3 Prototype Implementations

We have implemented the XAAL object hierarchy with a parser and a serializer. The implementation is on a prototype level, and not all elements are fully supported.

We have also implemented various adapters and generators between XAAL and other algorithm animation languages. Thus, we already have several different formats available for the same animation. The current selection of formats is presented

in Figure 3. In addition to the formats in the figure, the systems allow some other export formats as well. For example, ANIMAL can be used to export QuickTime movies and MatrixPro exports TeXdraw illustrations. Thus, we could create, for example, a QuickTime movie from an SVG animation (Scalable Vector Graphics) [11].

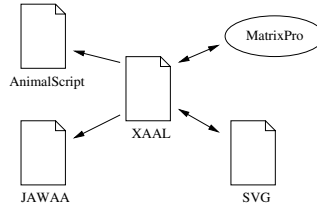


Fig. 3. Prototype format transformations implemented. The arrows represent the direction of the transformation. The ellipse represents an AV system, whereas the other are documents.

The prototypes do not implement all the features of the new language. For example, graphical primitive animation is currently not implemented, although it can be considered very important when exchanging information between systems like ANIMAL or JAWAA. Still, the prototypes enable us to transfer data between AV systems. Although the data currently is only static visualizations, we feel that we have demonstrated that this kind of approach is suitable for the problem at hand and it would be worthwhile to complete the implementation and include other languages in the future.

4 Discussion

In this paper, we have introduced an XML language for describing algorithm animations. The language is based on the specifications and examples of ITiCSE XMLWG. In addition, we have described a set of tools for exchanging data between algorithm animation systems. In the following, we will discuss what use this work can have for participants of the ITiCSE XMLWG, developers of AV systems, and teachers wanting to use AV.

For the participants and the continuing work of the working group, the specifications and tools presented in this paper provide concrete tools aiding the adoption of a common language. As the XML schema of XAAL is modular, the other participants of the group can take advantage of the specifications using the working group's definitions.

Developers of AV systems can implement different import and export formats for the existing systems with a reasonable effort. This can improve the applicability of the systems and thus promote the usage of AV systems.

Since the AV systems have different approaches to creating animations, an AV designer can combine the good features of multiple systems in the process of creating animations. This could lower the effort needed to create the animations.

Finally, for teachers, ready-made animations can be used in multiple systems and one system can use animations created for another system, thus expanding the selection of ready-made animations. In addition, the teacher can use only one system in teaching, thus helping students by providing animations with similar user

interface and appearance throughout a course. Furthermore, the teacher can select the system based on the level of interaction it provides.

4.1 Future Visions

We have numerous improvements and ideas for the future of the language, and here we will present some of the most interesting ones.

The most urgent requirement is to finish the prototype implementation of the parser and the adapters and generators. A natural continuation for this is to support new formats. These new formats could be other algorithm animation languages (for example, GAIGS-XML [7]), graph description languages (for example, GraphXML [2]), or something more different like OASIS OpenDocument, programmable graphics canvas element of HTML, or Macromedia Flash. Furthermore, as can be seen from Figure 3, many of the transformations are only available in one direction. A future challenge is to be able to generate XAAL documents with other systems, or alternatively, parse/transform other formats into XAAL.

The language could be extended to include programming concepts and thus allow the definition of algorithms and program visualization. There is also a need for more complete metadata and semantic information to be included in the algorithm visualizations allowing more flexible searching from algorithm animation repositories. This combined with a web-based service for transforming existing algorithm animations would lower the time needed for teachers to search for suitable ready-made examples and use existing animations in their teaching. Furthermore, we need to include and implement the interaction specification of the ITiCSE XMLWG.

For this to happen, it would be important for other educators to join in the development of common tools for the whole community. Having a group of researchers developing open source tools for AV systems could allow the development of high-quality tools easily adoptable by teachers.

References

- [1] Akingbade, A., T. Finley, D. Jackson, P. Patel and S. H. Rodger, *JAWAA: easy web-based animation from CS0 to advanced CS courses*, in: *Proceedings of the 34th SIGCSE technical symposium on Computer science education, SIGCSE'03* (2003), pp. 162–166.
- [2] Herman, I. and M. S. Marshall, *GraphXML - an XML-based graph description format*, in: *Graph Drawing*, 2000, pp. 52–62.
- [3] Hundhausen, C. D., S. A. Douglas and J. T. Stasko, *A meta-study of algorithm visualization effectiveness*, *Journal of Visual Languages & Computing* **13** (2002), pp. 259–290.
- [4] Karavirta, V., “XAAL - Extensible Algorithm Animation Language,” Master’s thesis, Department of Computer Science and Engineering, Helsinki University of Technology (2005), available online at <http://www.cs.hut.fi/Research/SVG/publications/karavirta-masters.pdf>.
- [5] Karavirta, V., A. Korhonen and L. Malmi, *Taxonomy of algorithm animation languages*, in: *Proceedings of the 2006 ACM symposium on Software visualization*, 2006, to appear.
- [6] Karavirta, V., A. Korhonen, L. Malmi and K. Stålnacke, *MatrixPro - A tool for on-the-fly demonstration of data structures and algorithms*, in: A. Korhonen, editor, *Proceedings of the Third Program Visualization Workshop*, Research Report CS-RR-407 (2004), pp. 26–33.
- [7] Naps, T., M. McNally and S. Grissom, *Realizing XML driven algorithm visualization*, in: *Proceedings of the Fourth Program Visualization Workshop*, 2006.


- [8] Naps, T. L., G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodgers 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, 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.
- [10] 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.
- [11] W3C, *Scalable Vector Graphics (SVG) 1.0 specification*, <http://www.w3.org/TR/SVG> (2001).

A Xaal Examples

```

1 <define-shape name="cat">
2   <circle-segment>
3     <center x="10" y="5"/>
4     <radius length="4"/>
5     <angle total="310" start="295"/>
6   </circle-segment>
7   <circle-segment>
8     <center x="10" y="15"></center>
9     <radius length="7"/>
10    <angle total="255" start="105"/>
11  </circle-segment>
12  ... <!-- specification of ears as lines and tail as arc -->
13 </define-shape>

```




Listing 1: Example of defining a shape, in this case, a cat.

```

1 <array indexed="false" size="7" orientation="horizontal">
2   <index index="0"><key value="A"/></index>
3   <index index="1"><key value="B"/></index>
4   <index index="4"><key value="C"/></index>
5 </array>

```

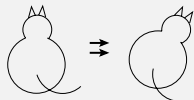


Listing 2: Example of an array definition.

```

1 <rotate degree="30" type="simple">
2   <object-ref id="catObj1"/>
3   <timing><delay s="2"/></timing>
4   <coordinate x="10" y="10"/>
5 </rotate>

```

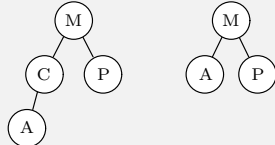


Listing 3: Example of a rotation operation. In the example, it is assumed that catObj1 is an instance of the shape defined in Listing 1.

```

1 <delete target="BST">
2   <key value="C"/>
3   <elementary>
4     <remove target="nodeC"/>
5     <remove target="edgeCA"/>
6     <replace target="edgeMC">
7       <edge from="nodeM" to="nodeA"/>
8     </replace>
9   </elementary>
10 </delete>

```



Listing 4: Example of a delete operation. The figures show the structure before (on the left) and after (on the right) the deletion.