Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 178 (2007) 129–135
www.elsevier.com/locate/entcs

Realizing XML Driven Algorithm Visualization

Thomas Naps[1]

Department of Computer Science
University of Wisconsin: Oshkosh
Oshkosh, WI, USA

Myles McNally[2]

Department of Mathematics and Computer Science
Alma College
Alma, MI, USA

Scott Grissom[3]

Department of Computer Science
Grand Valley State University
Allendale, MI, USA

Abstract

In this paper we describe work in progress on JHAVÉ-II, a new generation of the client-server based algorithm visualization system JHAVÉ. We believe this to be the first algorithm visualization system to be totally XML driven. We describe the XML scripting language visualization authors can use with JHAVÉ-II to define the sequence of graphical snapshots, integrated pop-up questions, synchronized pseudocode, and supplemental information that comprise a particular algorithm visualization. JHAVÉ-II then uses these scripts to render visualizations and support student exploration of algorithms.

Keywords: algorithm, visualization, xml

1 Introduction

Criteria for engaging students with an algorithm visualization (AV) have been previously detailed in [2]. These criteria are structured into a engagement taxonomy

[1] Email: naps@uwosh.edu
[2] Email: mcnally@alma.edu
[3] Email: grissom@gvsu.edu

1571-0661 © 2007 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2007.01.042

that includes *responding* (to questions about the visualization), *changing* (the visualization by providing new data to the algorithm being visualized), *constructing* (being responsible for the appearance of the graphical rendering done by the visualization), and *presenting* (using the visualization as a component of a written, oral, or hypertextual explanation of the algorithm). An effective instructional algorithm visualization must be much more than a sequence of pretty pictures. The complete instructional package in which AV is used must allow the visualization designer to ask questions of the learner, to strategically allow the learner to provide input to the algorithm, and to supplement the visualization with appropriate text such as synchronized pseudocode and other descriptions of the algorithm.

There is a tremendous amount of data that underlies effective AV – program state data, data set input generation, pseudocode, hypertextual explanations, and generation of non-trivial questions about the algorithm being viewed by the learner. Last year's ITiCSE working group on the "Development of XML-based Tools to Support User Interaction with Algorithm Visualization" recognized this fact. Its report [4] established a framework that defines a direction for future research and development, and raises a number of interesting issues in visualization system design. In this paper we describe a significant first step in addressing some of these issues by describing work in progress on what we believe to be the first instructional AV system based entirely on XML descriptions of underlying data. Although a first step, this system is much more than a prototype. Within the environment we have developed a substantial number of instructional visualizations, including a full collection of sorting, search-tree, and graph algorithms appropriate for a typical data algorithms and data structures course. By using XML as the underlying representation for the data manipulated by these visualizations, we have considerably reduced the amount of developer time required to produce a visualization. We have also better prepared ourselves for the inevitable extensions to the system that we will want to implement in the future.

2 JHAVÉ-II Architecture

The JHAVÉ-II architecture for delivering AV extends the original JHAVÉ AV environment described in [3]. As with the original JHAVÉ, JHAVÉ-II is still an Internet-based system that generates visualizations by executing algorithms on a server, generating a visualization script, and delivering that script to the JHAVÉ client for presentation to the learner. However, unlike the original JHAVÉ, JHAVÉ-II will rely upon XML as the language for defining these scripts.

In this architecture, the server application manages the available algorithms and generates the visualization scripts that the client displays. In a standard session, the learner first launches an instance of the client application, which displays a listing of available algorithms. When the user selects an algorithm from that list, the client sends a request to the server. The server knows what kind of input data the learner must provide for this algorithm and sends an description of an appropriate input generator object to the client. The client uses this to generate

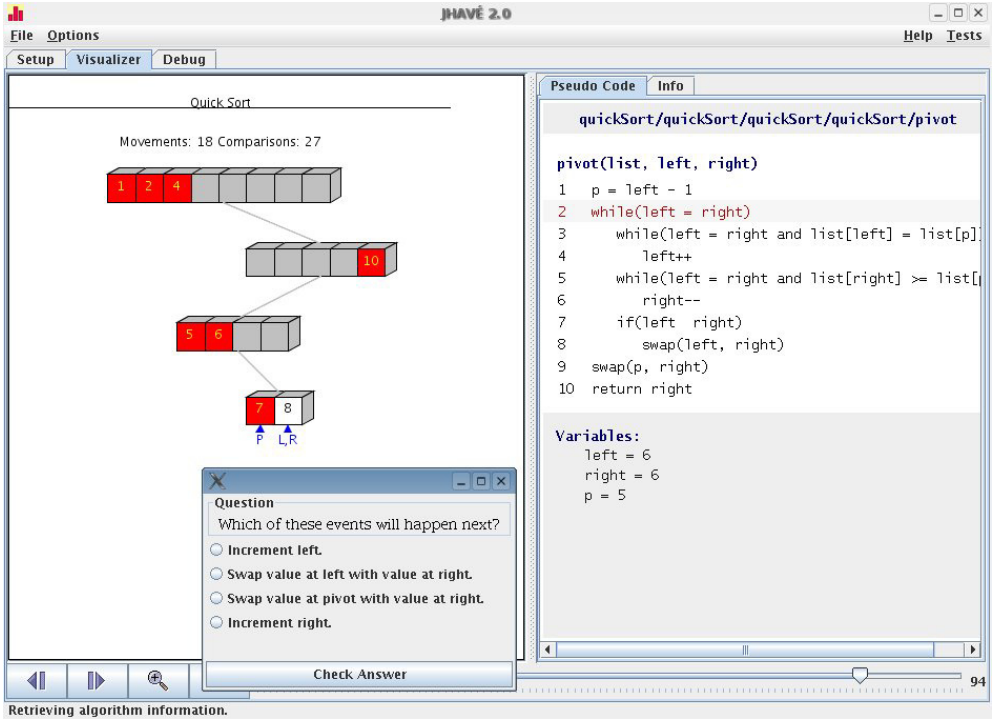

Fig. 1. JHAVÉ-II Screenshot

a frame with appropriate input areas for the learner. Once the learner fills out these areas, the client returns the input to the server as a data set to use when running the algorithm. The server then runs a program that generates the script for that algorithm and sends a URL back to the client from which the script can be read. The JHAVÉ-II client instantiates the appropriate visualizer plug-in to parse, render, and present the script to the learner – complete with a standard set VCR-like viewing controls, stop-and-think questions, and information/pseudocode windows. Figure 1 illustrates this for a visualization of the Quicksort algorithm with accompanying pseudocode window and a stop-and-think question.

The data flows during a JHAVÉ-II visualization session are depicted in Figure 2. Of these, the Vis Script flow is currently implemented in XML. This is the most complex of the flows and is partially described in the next section. The remaining flows (Choice, Input Generator, Input) remain work in progress and are not reported on here.

The original JHAVÉ client and the plug-ins it supported worked extremely hard at parsing the Vis Script received from the server. That was because, regardless of the plug-in used, the data came to the client in a relatively cryptic format that made sense only to someone familiar with the intricacies of the internals of JHAVÉ and the plug-in. In JHAVÉ-II, the XML data the client receives bears meaningful tags that clearly identify the various components of the script. Moreover, because of the broad range of the XML-parsing tools that are widely available, the client's paring of the data is a non-issue. The JHAVÉ-II client simply uses the JDOM

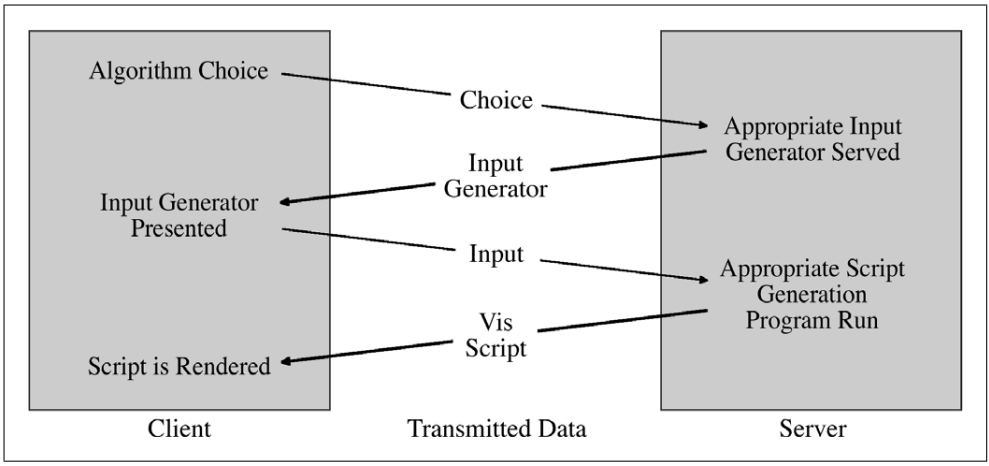

Fig. 2. JHAVÉ-II Data Flows

parser classes (available at http://www.jdom.org) to verify that it is receiving a syntactically valid script and then build an internal tree representation of the script. This internal tree is then recursively walked to render and present the visualization to the learner. Of the various visualizer plug-ins that were available for JHAVÉ (Animal[6], GAIGS[5], and Samba[7]), only GAIGS is currently supported for use with XML Vis Scripts. Animal support is in development.

3 XML Scripting for Visualization

In this section we provide brief descriptions of portions of the XML Vis Script language for JHAVÉ-II. It is essentially a scripting language that captures representations of data structures at the interesting events during an algorithm's execution. These snapshots of the data structures can then be augmented by the various supporting tools of the JHAVÉ-II environment. For each data structure capable of being described by the XML – stacks, queues, arrays, linked lists, trees, and graphs – we have also developed a class that implements the data structure along with a toXML method that can be used by the visualization designer to annotate an algorithm at an interesting event, thereby producing the XML necessary for the visualization script. The process of writing a visualization script-producing program is then to first implement the algorithm you wish to visualize and then annotate the program at its interesting events in a fashion similar to what one does when inserting tracer output to debug a program. The existence of the toXML methods for each data structure make it very painless to produce plain vanilla visualizations, to which can be added stop-and-think questions, synchronized pseudocode and documentation. Although production of such higher quality visualizations certainly requires more programming, we feel that the descriptive nature of the XML tags we use in our scripting language still make it a relatively painless process.

3.1 Overall Script Organization

The current plug-in for JHAVÉ-II specifies the XML for its scripts using XML DTD's (Data Type Definitions). For those not familiar with DTD's, an excellent description is given in [1]. In brief such a DTD resembles an Extended Backus-Naur Form (EBNF) description of a language. The DTD for JHAVÉ-II defined in Figure 3 specifies that a visualization is defined by a tagged entity called a show. Each such show consists of one or more snaps (that is, snapshots) followed by zero or more questions. A snap consists of a title, optional documentation and pseudocode URLs, zero or more data structures (stacks, queues, arrays, linked lists, trees, or graphs), and an optional question-reference for the snapshot.

```
< !ELEMENT    show    ( snap+, questions?) >
< !ELEMENT    snap    ( title,
                        doc_url?
                        pseudocode_url?,
                        ( tree | array | graph | stack | queue |
                          linkedlist | bargraph | node )*,
                        question_ref? ) >
```

Fig. 3. High Level Script DTD

3.2 Data Structures

Each of the six data structures that can be rendered by the current JHAVÉ-II plug-in has its own DTD definition. As an example, consider the stack definition in Figure 4. The data in a stack consists of a sequence of zero or more list items. The optional bounds tag may be used to specify the position and size of the stack picture that is rendered by the plug-in. The color attribute for a list_item is used to specify the color of each data item in the stack.

```
< !ELEMENT    stack       ( name?, bounds?, list_item*) >
< !ELEMENT    bounds      ( EMPTY ) >
< !ATTLIST    bounds      x1 CDATA #REQUIRED
                          y1 CDATA #REQUIRED
                          x2 CDATA #REQUIRED
                          y2 CDATA #REQUIRED
              fontsize    CDATA "0.03" >
< !ELEMENT    list_item   ( label )>
< !ATTLIST    list_item   color CDATA "#FFFFFF" >
< !ELEMENT    label       ( #PCDATA ) >
```

Fig. 4. Stack Data Structure DTD

Of course, non-linear data structures such as trees and graphs have more complicated DTD definitions. Nonetheless the bounds tag and color attribute are used in a consistent fashion throughout all of the data structure definitions.

3.3 Documentation, Pseudocode, and Interactive Questions

The support offered by JHAVÉ-II for its plug-ins includes documentation and pseudocode windows. The DTD for documentation window content is merely a reference to a URL that specifies an HTML document. Pseudocode windows are more complicated as they must be synchronized with the state of the data structure that is being viewed by the learner. For example, Figure 1 shows a pseudocode window for a visualization of the quicksort algorithm. In addition to the program listing, note the call stack and the current values of individual variables. The DTD for such a pseudocode window appears in Figure 5.

```
< !ELEMENT   doc_url          ( #PCDATA ) >
< !ELEMENT   pseudocode       ( call_stack?,
                                program_listing?,
                                variables? ) >
< !ELEMENT   call_stack       ( #PCDATA ) >
< !ELEMENT   program_listing  ( signature?, line* ) >
< !ELEMENT   signature        ( #PCDATA ) >
< !ELEMENT   line             ( ( #PCDATA | replace ) + ) >
< !ATTLIST   line             line_number CDATA #IMPLIED>
< !ELEMENT   variables        ( variable* ) >
< !ELEMENT   variable         ( #PCDATA, replace ) >
< !ELEMENT   replace          ( EMPTY ) >
< !ATTLIST   replace          var NMTOKEN #REQUIRED >
```

Fig. 5. Pseudocode DTD

A DTD for interactive stop-and-think questions has also been defined. Presently four types of questions are supported – true-false, fill in the blank, multiple choice, and multiple selection (multiple choice with more than one right answer).

4 Conclusions

In this paper we have described work in progress on JHAVÉ-II, the next generation of the client-server based JHAVÉ AV environment. While this new release will have a number of enhancements, we focused here on the conversion of all JHAVÉ data flows to the XML format. Positive outcomes of this conversion will include enhanced extensibility and ease of visualization development.

5 Acknowledgement

This work was supported by a United States National Science Foundation CSLI Grant, DUE-0126494.

References

[1] Harold, E. and S. Means, "XML in a Nutshell," O'Reilly, 2004.

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

[3] Naps, T., J. Eagan and L. Norton, *Jhavé – an environment to actively engage students in web-based algorithm visualizations*, ACM SIGCSE Bulletin **32** (2000), pp. 109–113.

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

[5] Naps, T. and B. Swander, *An object-oriented approach to algorithm visualization - easy, extensible, and dynamic*, ACM SIGCSE Bulletin **26** (1994), pp. 46–50.

[6] Rößling, G. and B. Freisleben, *Animalscript: An extensible scripting language for algorithm animation*, ACM SIGCSE Bulletin **33** (2001), pp. 70–74.

[7] Stasko, J., *Using student-built algorithm animations as learning aids*, ACM SIGCSE Bulletin **29** (1997), pp. 25–29.