USING STRUCTURED HYPERMEDIA TO EXPLAIN ALGORITHMS

Tomasz Müldner*, Elhadi Shakshuki*, Andreas Kerren[+], Zhinan Shen*, Xiaoguang Bai*
**Jodrey School of Computer Science, Acadia University*
Wolfville, NS, Canada B4P 2R6
{Tomasz.Muldner, Elhadi.Shakshuki, Zhinan.Shen, Xiaoguang.Bai}@acadiau.ca

[+] *Computer Science Department, University of Kaiserslautern*
P.O.Box 3049, D-67653 Kaiserslautern, Germany
kerren@informatik.uni-kl.de

ABSTRACT

Most systems designed to teach algorithms using visualization and animation techniques have not proved to be *educationally effective*. To satisfy this aim, some recently built systems use a *hypermedia environment* to provide knowledge and context to explain algorithms. In this paper, we describe a system called Structured Hypermedia Algorithm Explanation (SHALEX), which provides several novel and important features. In particular, our hypermedia environment can reflect the structure of an algorithm. We define this structure as a directed graph of abstractions, where each abstraction is designed to *focus* on a single operation used directly or indirectly in the algorithm. This way an algorithm may be studied top-down, bottom-up, or using a mix of the two. In addition, SHALEX includes a student model to provide spatial and temporal links, and to support evaluations and adaptations.

KEYWORDS

Algorithm explanation, visualization, hypermedia, computer science education

1. INTRODUCTION

Because of the importance of algorithms, for more than 15 years researchers have been trying to find the best way to teach algorithms. Visualization is considered as one of the best known approaches to use; especially its understanding as “the power or process of forming a mental image of vision of something not actually present to the sight”, see Petre et al. (1998a). Software visualization has grown from early flow-charts to current sophisticated graphics workstations showing 3D visualization of complex software systems. It uses various kinds of *multimedia*, including graphics to show abstractions of data, animation and video to convey the temporal evolution of a computer algorithm, see Stasko & Lawrence (1998) and voice, also called auralization, see Brown & Hershberger (1998). There have been many papers describing the use of animation to software explanation. In this paper, we will not review these usages and interested readers are referred to Gloor (1992), Gloor (1998) and Kerren & Stasko (2001). The most recent overview evaluation of the educational effectiveness of algorithm visualization is given in Hundhausen et al. (2002). Evaluations of systems designed to teach algorithms using various visualization and animation techniques have not shown that these systems are educationally effective. Indeed, some studies found that the effect of using animation is either neutral or negative, see Stasko & Lawrence (1998). In this paper, by the *algorithm explanation system* we mean a system designed to teach algorithms, using multimedia, including but not limited to graphics and animation.

This paper extends ideas that are originally presented in Müldner (2003), and then in Müldner & Shakshuki (2004a, 2004b), Müldner, Shakshuki & Merill (2004a, 2004b), and Shakshuki, Müldner & Haughn (2004). Here, we describe our algorithm explanation system, called Structured Hypermedia Algorithm Explanation (SHALEX). This system includes a hypermedia environment which provides links between various kinds of multimedia. Unlike existing algorithm explanation systems, SHALEX reflects the

Proceedings of the IADIS International Conference e-Society '05,
Qawra, Malta, 2005.

structure of an algorithm. We define this structure as a directed graph of abstractions. Each abstraction is designed to *focus* on a single operation used directly or indirectly in the algorithm, and it provides an Abstract Data Type, ADT, which gives a high-level view of generic data structures and operations. Operations are provided in a textual form, but there is also a hyperlinked visual description used to help the student to understand basic properties of the algorithm; for example *algorithm invariants*. Each ADT operation is either implemented in an abstraction at the lower level, or it is a *primitive* operation. This approach supports the novel mode of studying not available in any other visualization system; namely an algorithm may be studied *top-down*, *bottom-up*, or using a *mix* of the two.

The second weakness of existing systems designed to teach algorithms is that they do not address the issue of implementing an algorithm in a specific programming language, or finding the time complexity of the algorithm. Our proposed system provides tools designed to tackle both these issues. The third major weakness of existing systems is that they do not *adapt* to the students behaviour. Therefore, a good student may be bored while a novice student may be overwhelmed. SHALEX includes a student model to provide spatial and temporal links, and to support evaluations and adaptations.

SHALEX is being implemented in Java, using XML. Java is used to implement the basic functionality and GUIs. XML is used to represent system data (such as all algorithms, all users, nodes of AAM for specific algorithms, etc.) as well as the author model and a student model. XML data are made persistent using a native XML database, eXist, see Exist database (2005).The advantage of using XML is that the student and author interface provide a view of various data using preferred format. For example, when the student requests the HTML view of all algorithms available in SHALEX, then the XML data are transformed using XSLT to HTML and displayed. The entire system is "designed for change", e.g. both models can be plugged into the system without making any changes in the system's architecture.

The rest of this paper is organized as follows. In Section 2, we briefly discuss related work on algorithm visualization and animation. Then, in Section 3, we describe our approach, followed by some conclusions and our future research in Section 4.

2. RELATED WORK

Using the standard approach to algorithm visualization, the user has to map the problem domain (values to be sorted) to the graphical domain (bars), and then looking at the animation the user has to retrieve essential properties of the algorithm (such as maintaining a sorted prefix). Therefore, this and many other existing algorithm animation systems resemble visual debuggers in that they show the *execution* of the algorithm by code-stepping, work at the lowest level of abstraction, and illustrate only the primitive code. This approach constrains users to view the code in the order of execution, which is the wrong information for understanding the algorithm and has a poor cognitive fit with the plan-and-goal structures that users are trying to extract from the code (see Petre et al. 1998a). In any case, runtime interpretation requires specific input data and cannot consider all possible inputs and often suffers from the lack of focus on relevant data (see Braune & Wilhelm 2000).

Recently, researchers have attempted to use a *hypermedia environment* to provide knowledge and context to explain algorithms. The most notable example of this approach is HalVis (see Hansen et al., 2002), which showed the advantage of using hypermedia over using just animations. The same paper argues that an algorithm is a process that is both abstract and dynamic, and a system designed to explain algorithms should emulate both these features. Since SHALEX extends this work, for the sake of completeness, below we briefly summarize most important features of HalVis:

- support for enhanced learning, with interactive examples, which helps students to understand what the algorithm is doing and why;
- support for active learning, by providing various kinds of questions (note that HalVis does not evaluate student's answers);
- hyperlinks that help the learner to move between various kinds of descriptions, e.g. text and animations;
- providing the analogical animation, and micro- and macro-animations.

Some systems show both, the animation and the pseudocode for the algorithm; for example, the Ganimal system, see Ganimal (2002) and Diehl & Kerren (2002). This system displays an abstract syntax tree of a programming language that extends Java and does not use pseudocode.

3. STRUCTURED HYPERMEDIA ALGORITHM EXPLANATION

This section describes our proposed system, which we call Structured Hypermedia Algorithm Explanation (SHALEX). This system is more general than HalVis, and it provides a number of additional and important features. Users of SHALEX can play one of the following three roles: *administrators*, who are responsible for maintaining user accounts; *authors*, who are responsible for creating algorithm explanations, various lessons, assigning evaluations, etc. – see section 3.1; and *learners*, who study algorithms.

3.1 Key Features

(1) *Structured hypermedia, which reflects the structure of algorithms.*

In SHALEX, operations are provided in a textual form, but there is also a hyperlinked visual description used to help the student to understand basic properties of an algorithm; for example algorithm invariants. Each operation is either implemented in an abstraction at the lower level, or it is a primitive operation. This is a generalization of micro/macro-level animations used in HalVis, which will allow the novel mode of studying unavailable in any other visualization system; namely, an algorithm may be studied top-down, bottom-up, or using a mix of the two (for more details see (2) below).

We define the algorithm structure as a directed graph of abstractions. Each abstraction is designed to *focus* on a single operation used directly or indirectly in the algorithm, and it provides an ADT, which gives a high-level view of generic data structures and operations. A hierarchical Abstract Algorithm Model (AAM) is a tree, which consists of *abstractions* representing operations. Each abstraction explains a single operation op(), and consists of a textual representation and a visual representation. The textual representation includes an ADT that provides data types and operations. It also provides a representation of the operation op() using the ADT from this abstraction. To explain this part, let's assume that f() is an operation. The abstraction that explains f(), abst(f) is a pair (ADT, representation of f() in the ADT), where ADT consists of data types and primitive operations. There is an edge from the abstraction abst(f) to an abstraction abst(g) if g is one of the primitive operations from the ADT abst(f). Therefore, a child abstraction provides a partial implementation of the operation from the parent abstraction. Typically, there are only few operations from any abstraction's ADT that are implemented in a child of this abstraction; others are considered *primitive* operations. An AAM of an algorithm f() is a tree rooted at abst(f). To explain an algorithm, we construct an AAM tree with sufficient number of levels so that the student is able to understand how and why the algorithm works (in particular, the student can form and justify invariants of the algorithm.)

Various examples of abstractions and algorithm explanations are provided in Müldner (2003), Müldner & Shakshuki (2004a, 2004b), Müldner et al. (2004a, 2004b).

A *linked* visual representation may be used by the student to help him or her understand the basic properties of this abstraction, such as invariants of the selection sort. Additional hyperlinks provide a description of fundamental concepts and an intuitive analogy (as in HalVis), and temporal and spatial links based on the student model (see below).

(2) *Active learning.*

SHALEX is an interactive system, which allows the student to select one of the available algorithms to study, and then it uses *a student model* to record student activities. These interactions are vital to support the active learning model. SHALEX helps the student to understand both *what* the algorithm is doing and *how* it works; as well *why* the algorithm works (algorithm correctness). In addition, SHALEX uses an *author model* to record decisions made by an author. For example, the author may decide to prepare, for a single algorithm, various *lessons* with different evaluations, various AAM trees providing more or fewer abstractions, etc.

Authors' responsibilities include creation of measures to evaluate the student performance. These measures include time, the percentage of questions that are corrected answered by student, etc. The author model may also include assignments of various *skill levels* to the student. If this is the case, then there will be two types of evaluation; to decide whether the student's skill level should be changed, and to decide whether the student has successfully learned the operation in question.

(3) *Support for programming the algorithm in any procedural programming language.*

SHALEX provides the intermediate representation of all AAM's primitive operations, called an Abstract Implementation Model (AIM). To implement the algorithm in a specific programming language, the student has to map to the selected language all primitive operations that do not have implementations in the AAM. The representations in AIM are *generic* in that they are not using any specific programming language; instead they use high-level concepts that can be mapped to many procedural programming languages.

(4) *Support for understanding time complexity of the algorithm.*

Explanation of algorithm complexity is one of the most difficult goals of algorithm visualization, because it requires mathematical proofs that are hard to visualize. The only attempt in this direction we are aware of is described in Pape & Schmitt (1997). The current version of SHALEX includes three kinds of tools designed to help the student to derive the complexity of the algorithm being studied. The first tool, based on Horstmann (2001), which gives the student a chance to experiment with various data sizes and plot a function that approximates the time spent on execution with these data. The second tool, based on Goodrich & Tamassia (2001), which provides visualization that helps to carry out time analysis of the algorithm. Finally, the third tool asks students various questions regarding the time complexity, and questions specific to the algorithm being studied, and evaluates their answers.

4. CONCLUSIONS AND FUTURE RESEARCH

This paper presented our proposed system for explaining algorithms, which is based on structured hypermedia approach. It has been shown that the proposed system has some fundamental advantages, including availability of studying an algorithm top-down, bottom-up, or using a mix of the two; support for understanding invariants; building a student model to provide spatial and temporal links; and the use of XML to store information. Our future work includes a generalization of AAMs to more general knowledge graphs, which consist of knowledge units. Then, AAMs or precedence graphs (see Muldner & Tan 1995) would become specialized knowledge graphs, and most of the general functionality would be available to both specific models. This will make it unnecessary to re-implement the graph from scratch.

The first version of algorithm visualization system was implemented using Macromedia Flash (2003). For the next version, we are considering using a HTML page to display our visualization (this design follows the design of Ganimal).

ACKNOWLEDGEMENT

This project is supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) and an internal grant from research and graduate studies of Acadia University. The authors would like to thank two alumni Jodrey School of computer science for their contributions to this project; Joe Merrill for his implementations in Macromedia Flash MX during the first stages of the project, and Brad Haughn for his implementations of some parts of the proposed system.

REFERENCES

Braune, B., & Wilhelm, R. (2000). Focusing in Algorithm Explanation. *IEEE Transactions on Visualization and Computer Graphics* 6(1), pp. 1-7.

Brown, M.H., & Hershberger, J. (1998). Program Auralization. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price (Eds.), *Software Visualization: Programming as a Multi-Media Experience*, MIT Press, pp. 137-144.

Diehl, S. & Kerren, A. (2002). Reification of Program Points for Visual Execution. In Proceedings of the First IEEE *International Workshop on Visualizing Software for Understanding and Analysis*, VisSoft 2002, IEEE Computer Society Press, pp. 100-109.

Exist database (2005). http://exist.sourceforge.net/

Ganimal. Project Homepage (2002). http:://www.cs.uni-sb.de/GANIMAL

Gloor, P. A., (1992). AACE - Algorithm Animation for Computer Science Education. Proceedings of the 1992 IEEE Workshop on Visual Languages, Seattle, WA, pp. 25–31.

Gloor, P. A., (1998). Animated Algorithms. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price (Eds.), *Software Visualization: Programming as a Multi-Media Experience*, MIT Press, pp. 409-416.

Goodrich, M. and Tamassia, R., (2001). *Data Structures and Algorithms in Java*. John Wiley & Sons, Inc., 2nd edition.

Hansen, S. R., Narayanan, N. H. & Hegarty, M. (2002). Designing educationally effective algorithm visualizations: Embedding analogies and animations in hypermedia. Journal of Visual Languages and Computing, 13(3), Academic Press, pp. 291-317.

Horstmann, C. (2001). *Big Java: Programming and Practice*. John Wiley & Sons.

Hundhausen, C.D., Douglas, S.A., & Stasko, J.T. (2002). A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages and Computing 13(3)*, pp. 259-290.

Kerren, A., & Stasko J.T. (2001). Algorithm Animation. Springer LNCS issue on Software Visualization. S. Diehl (Ed.): Software Visualization International Seminar, Dagstuhl Castle, Germany. Revised Papers LNCS 2269, Springer, pp. 1-15.

Macromedia Flash MX (2003). http://www.macromedia.com/software/flash/.

Müldner, T. & Shakshuki, E. (2004a). A new Approach to Learning Algorithms. Proceedings of the International Conference on Information Technology (ITCC2004), IEEE Computer Society, Las Vegas, USA, pp. 141-145.

Müldner, T. & Shakshuki, E. (2004b). On Visualization and Implementation of Algorithms. The 2004 International Conference on Modeling, Simulation and Visualization Methods (MSV'04), Las Vegas, USA, pp. 136-141.

Müldner, T. & Tan, L.B. (1995). Generating Individualized Curricula. EDMEDIA'95, Graz, Austria.

Müldner, T. (2003). An Algorithm for Explaining Algorithms. November 2003, University Technical report TR-2003-01. http://cs.acadiau.ca/technicalReports/

Müldner, T., Shakshuki, E. & Merill J. (2004b). Selecting Media for Explaining Algorithms. AACE Proceedings of EDMEDIA'04; Lugano, Switzerland, pp. 2048-2053.

Müldner, T., Shakshuki, E. & Merill, J. (2004a). Yet another Way to Explain Algorithms. The 7th IASTED International Conference on Computers and Advanced Technology in Education, CATE 2004, Hawaii, USA, pp. 136-141.

Pape C. & Schmitt P.H. (1997). Visualizations for Proof Presentation in Theoretical Computer Science Education. In Z. Halim, Th. Ottmann, and Z. Razak, editors, Proceedings of International Conference on Computers in Education, AACE - Association for the Advancement of Computing in Education, . pp. 229–236.

Petre, M., Baecker, R, & Small, I. (1998a). An Introduction to Software Visualization. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price (Eds.), *Software Visualization: Programming as a Multi-Media Experience*, MIT Press, pp. 3-26.

Shakshuki, E., Müldner, T. & Haughn, B. (2004). ATIA: Algorithm Teaching Interface Agent. Proceedings of the 5th International Conference on Information Technology Based Higher Education and Training (ITHET'04), IEEE Computer Society, pp. 138-143.

Stasko, J., & Lawrence, A. (1998). Empirically Assessing Algorithm Animations as Learning Tools. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price (Eds.), *Software Visualization: Programming as a Multi-Media Experience*, MIT Press, pp. 417-419.