Web-based Structured Hypermedia Algorithm Explanation system

Elhadi Shakshuki
Jodrey School of Computer Science, Acadia University, Wolfville, Canada
Andreas Kerren
School of Mathematics and Systems Engineering, Växjö University, Växjö, Sweden, and
Tomasz Müldner
Jodrey School of Computer Science, Acadia University, Wolfville, Canada

Abstract

Purpose – The purpose of this paper is to present the development of a system called Structured Hypermedia Algorithm Explanation (SHALEX), as a remedy for the limitations existing within the current traditional algorithm animation (AA) systems. SHALEX provides several novel features, such as use of invariants, reflection of the high-level structure of an algorithm rather than low-level steps, and support for programming the algorithm in any procedural or object-oriented programming language.

Design/methodology/approach – By defining the structure of an algorithm as a directed graph of abstractions, algorithms may be studied top-down, bottom-up, or using a mix of the two. In addition, SHALEX includes a learner model to provide spatial links, and to support evaluations and adaptations.

Findings – Evaluations of traditional AA systems designed to teach algorithms in higher education or in professional training show that such systems have not achieved many expectations of their developers. One reason for this failure is the lack of stimulating learning environments which support the learning process by providing features such as multiple levels of abstraction, support for hypermedia, and learner-adapted visualizations. SHALEX supports these environments, and in addition provides persistent storage that can be used to analyze students' performance. In particular, this storage can be used to represent a student model that supports adaptive system behavior.

Research limitations/implications – SHALEX is being implemented and tested by the authors and a group of students. The tests performed so far have shown that SHALEX is a very useful tool. In the future additional quantitative evaluation is planned to compare SHALEX with other AA systems and/or the concept keyboard approach.

Practical implications – SHALEX has been implemented as a web-based application using the client-server architecture. Therefore students can use SHALEX to learn algorithms both through distance education and in the classroom setting.

Originality/value – This paper presents a novel algorithm explanation system for users who wish to learn algorithms.

Keywords Interactive devices, Multimedia systems, Programming, Programming languages, Learning methods

Paper type Research paper

The first and the third authors worked on this project with a partial support provided by a grant from the Natural Sciences and Engineering Research Council of Canada and internal grants from the Teaching and Innovation fund, Acadia University. The authors would like to thank two alumni of 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.

International Journal of Web Information Systems
Vol. 3 No. 3, 2007
pp. 179-197
© Emerald Group Publishing Limited
1744-0084
DOI 10.1108/17440080710834238

Introduction

The analysis and the understanding of algorithms is a very important task for teaching and learning algorithms. We advocate a strategy, according to which one first tries to understand the fundamental nature of an algorithm, and then – after reaching a higher level of awareness – chooses the most appropriate programming language to implement it. To facilitate the process of understanding of algorithms, their visualization, in particular animation is considered to be the best approach. This approach is described in the next subsection.

Algorithm animation

An algorithm animation (AA) visualizes the behaviour of an algorithm by producing an abstraction of both the data and the operations of the algorithm. At first, it maps the current state of the algorithm into a picture which is then animated based on the change between two succeeding states of the running algorithm. This way, AA facilitates better understanding of the inner workings of the algorithm. Specifically, it reveals algorithm's deficiencies and advantages, thereby allowing further optimization (Gloor, 1992, 1998a). Price *et al.* (1993) distinguished between AA and program animation. The former term refers to a dynamic visualization of the higher-level descriptions of software (algorithms) that are later implemented in software. The latter term refers to the use of dynamic visualization techniques to enhance human understanding of the actual implementation of programs or data structures. They defined both areas of study to collectively be a part of Software Visualization (SV).

Many researchers have attempted to describe the development and use of AA. For more details and an overview of AA tools, interested readers are referred to the introduction by Kerren and Stasko (2002) in the book (Diehl, 2002). Furthermore, two extensive anthologies on SV providing overviews of the field were published in 1996 and 1998 (Eades and Zhang, 1996; Stasko *et al.*, 1998). The latter anthology contains revised versions of some seminal papers on classical AA systems as well as educational and design topics. Other published articles provide summaries of different aspects of AA, including taxonomies (Brown, 1988), the use of abstraction (Cox and Roman, 1992), and user interface issues (Gloor, 1998b).

Drawbacks of traditional systems

Evaluations of systems designed to explain algorithms using various visualization and animation techniques have not shown that these systems are educationally effective (Hundhausen *et al.*, 2002). However, software evaluations are difficult to verify and widely used test designs have various disadvantages (Baumgartner, 1999). If we agree that weak evaluation results are true and significant then we have to look for reasons to prevent such results in the future.

One reason of a failure could be that many existing AA 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 statements. This approach constrains users to view the code in the order of execution, which is the wrong information for understanding the algorithm. It has a poor cognitive fit with the plan-and-goal structures that users are trying to extract from the code (Petre *et al.*, 1998a). Furthermore, runtime interpretation requires specific input data and cannot consider all possible inputs and often suffers from the lack of focus on relevant data

(Braune and Wilhelm, 2000). A related problem is the missing representation of algorithm invariants in most AA systems. Existing systems do not address the issue of implementing algorithms in specific programming languages, paying attention to their structure, or finding their time complexity. Adapting facilities for the learner behaviour are not supported, nor is the additional use of media beyond graphics and animation.

The remainder of this paper is organized as follows. Second section provides an outline of several well-known algorithm explanation methods. Third section describes our approach and provides its most important features and implementation aspects. Authoring algorithm explanations and learning tasks are briefly exemplified in fourth section. Lastly, fifth section concludes the paper and highlights our future work.

Explanation methods

By an algorithm explanation system we mean a system designed to teach algorithms using multimedia that includes but is not limited to graphics and animation. There are various existing approaches to explain algorithms. All approaches including visualization, abstraction, constructivism and hypermedia have their specific advantages and problems. These approaches are briefly described in the following paragraphs.

Visualization techniques

Static visualizations (such as flowcharts) and dynamic AAs are the most popular way to explain their design and behaviour. As we mentioned above, this and many other existing AA systems resemble visual debuggers. The runtime interpretation requires specific input data and cannot consider all possible inputs and often suffers from the lack of focus on relevant data (Braune and Wilhelm, 2000). One particular problem with the dynamic execution of the algorithm is that the user has to remember the "previous state". Multiple views showing algorithm states are used to avoid forcing the viewer to remember the previous states (Biermann and Cole, 1999). The JHAV'E system (Naps, 2005) is a support environment for a variety of available AA systems. It provides several interaction support tools, such as input generators, stop-and-think questions, VCR controls, etc.

Abstraction

Algorithms represent abstract processes but this aspect is rarely considered. One approach presented by Wilhelm *et al.* (2002) uses a static source code analysis to abstractly execute the algorithm on "all possible sets of input data" and visualize invariants. An extension of this approach was exemplified for binary tree algorithms (Johannes *et al.*, 2005). The idea of using multiple levels of abstraction is supported by Petre *et al.* (1998b) who claim that in general it is hard to determine a single suitable level of abstraction. Their research has shown that if the presentation is designed to highlight some kind of information, then it is likely to obscure other kinds. In our approach, each level of abstraction is used to highlight a single kind of information, for example invariants. So, the learner can focus on this kind of information.

The abstract model of the algorithm often uses pseudocode and it includes the high-level abstract data structures and operations. These operations are designed so that they can be directly mapped to most procedural and object-oriented programming languages. Using pseudocode, the algorithm can be studied independently of any programming language (Fleischer and Kucera, 2002; Naps, 2005). The pseudocode may have an additional visual representation which exposes its properties, in particular its invariants.

Concept keyboards

Baloian *et al.* (2005) suggested using so-called concept keyboards (CKs) in order to explore data structures and to execute the methods of an algorithm. Each key of a CK is mapped to the execution of an existing method available in the implementation of the input algorithm. Based on the offered keys, the user can trigger more complex or abstract operations. The approach does not focus on visualizations themselves: visualizations or other media (sound or movies) are to only reflect the users' attempts at algorithms and data structures. Several evaluations show that the active use of CKs leads to a better understanding of how algorithms work. Our approach has some similarities with algorithm visualization using CKs. Hypermedia including visualization is used in our system to reflect the current information. The main difference is that we use a flexible graph structure for an algorithm to describe operations and their dependencies.

Constructivism

The constructivist approach is based on the idea that the knowledge has to generate itself in the learner's mind. Therefore, knowledge cannot be transferred in a traditional way, e.g. by instruction. Within the moderate constructivism, the teacher, the expert and the system are not allowed to manipulate the learner's construction process but they can offer help and coach their individual construction processes. Therefore, a goal of the moderate constructivism is to build learning environments that give learners the possibility to generate their own knowledge constructs. One possibility to reach this goal in the context of learning algorithms is to use compiler generation techniques to generate interactive AA from specifications (Kerren, 2004a, b).

Constructivism principles are used in active learning (Hundhausen *et al.*, 2002) and this style of learning includes various kinds of interactions with the learner. For example, students are able to use their own input data sets; use a do-it-yourself mode and predict the next step of the algorithm, or determine the essential algorithm properties. Enhancing this idea, algorithm explanations should not be prepared by experts; instead they should be prepared by learners themselves. Additionally, they should support programming the target algorithm, using a standard programming language. This ability is missing from all existing systems, but in our opinion it is absolutely essential.

Hypermedia

Development of hypermedia environments to provide knowledge and context to explain algorithms is a relative new research area. The most notable example of this approach is HalVis (Hansen *et al.*, 2002), which showed the advantage of using hypermedia over using just animations. The authors argue 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, Structured Hypermedia Algorithm Explanation (SHALEX) extends this work, we briefly summarize several most important features of HalVis: support for enhanced learning with interactive examples which helps learners 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 learner's answers); hyperlinks that help the learner to move between various kinds of descriptions, e.g. text and animations; and finally the analogical animation, including both, micro and macro-animations.

Although HalVis is a very useful system and is one of the few systems that provide hypermedia, it has several serious limitations. For example, HalVis only allows the users to learn in one direction using a top-down approach, which does not always reflect the structure of the algorithm and is not adaptive. Additionally, it supports abstractions, but only for micro/macro-level animations.

Another AA system, called Ganimal (Diehl and Kerren, 2002; Ganimal, 2007), supports hypermedia as follows: all algorithms are implemented in an AA specification language Ganila. Ganila offers a set of control structures, such as the possibility to annotate the statements of the underlying algorithm with URLs. Ganila programs are translated into Java and executed within an own runtime system for animation. If the system performs an annotated statement then a HTML-View is opened. This view can interpret pure HTML code, show images, foreign Java applets, Flash animations, etc. to support the learning process. Furthermore, it is possible to play sound if a special program point is executed. Ganimal does not support abstraction levels or learner evaluations, but it is a powerful system to produce stand-alone hypermedia animations.

Structured hypermedia algorithm explanation

This section discusses our algorithm explanation system that includes a hypermedia environment providing links between various kinds of multimedia. Our system, called SHALEX (2007), aims to address most of the aforementioned problems of systems described in the first section. The most novel property of SHALEX, which makes it possible to reach this ambitious goal, is that it reflects the structure of an algorithm, defined as digraph of abstractions. Thus, it is possible to support several levels of abstractions which help the learner to understand basic properties of the algorithms as well as to recognize good implementation strategies.

Concepts and features

A major weakness of many existing systems is that they do not adapt to the learner's behaviour. Therefore, a good student may be bored while a novice student may be overwhelmed. SHALEX includes a learner model to provide spatial and temporal links, and to support evaluations and adaptations. In this context, the system's users can play one of the following four roles:

(1) learners (students), who study algorithms;

(2) authors, who are responsible for tasks such as creating algorithm explanations, various lessons, or assigning evaluations;

(3) administrators, who are responsible for tasks such as maintaining user accounts and their roles; and

(4) algorithm administrators, who are responsible for tasks such as group management of users assigned to study specific algorithms, management of algorithm explanations, including log information.

SHALEX supports many algorithms; explanations of which are created by various authors. To support this, we designed a taxonomy of explanations which has a tree-like structure. Non-leaf nodes of the taxonomy represent concepts, such as "Iterative Algorithms" (the root represents all algorithms). Leaves represent explanations of specific algorithms, created by specific authors, for example "John Doe: Merge Sort". The author who creates an explanation of a new algorithm specifies where in the taxonomy hierarchy this explanation will be placed (Figure 5, upper screenshot).

Structured hypermedia and abstraction levels. In our approach, operations are provided in a textual form, but there is also a hyperlinked visual description used to help the learner 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/macrolevel animations used in HalVis (Hansen *et al.*, 2002) which will allow the novel mode of studying unavailable in any other visualization system: an algorithm may be studied top-down, bottom-up, or using a mix of the two (for more details see below).

We define the algorithm structure as a hierarchical Abstract Algorithm Model (AAM) which is a directed acyclic graph with nodes representing abstractions and directed edges representing operation dependencies. Each abstraction is designed to focus on a single operation used directly or indirectly in the algorithm, i.e. it explains a single operation op and consists of a textual representation and a visual representation. The textual representation includes, among other things, an abstract data type (ADT) that gives a high-level view of generic data structures and operations.

Let as assume that f is an operation. The abstraction that explains f, abst(f) is a pair (ADT, repr(f)), where ADT consists of data types and primitive operations (Figure 1). There is a directed edge from the abstraction abst(f) to an abstraction abst(g) if g is one of the primitive operations from the ADT abst(f). Thus, a successor abstraction provides a partial implementation of the operation from the predecessor abstraction. Typically, there are only few operations from any abstraction's ADT that are implemented in a successor of this abstraction; others are considered primitive operations. An AAM of an algorithm f is a graph sourced at abst(f).

To build an algorithm explanation, we construct an AAM with a sufficient number of levels so that the learner is able to understand how and why the algorithm works. In particular, the learner can form and justify invariants of the algorithm. Let us consider the insertion sort algorithm (Aho *et al.*, 1983) as an example. Each iteration of this algorithm removes an element from the input data, inserting it at the correct position in the already sorted list until no elements are left in the input. Insertion sort can be implemented using operations from two ADTs: the insertion ADT provides

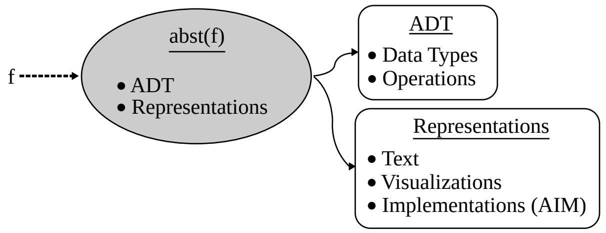


Figure 1.
Abstraction node of the AAM

generic operations, such as insert and the primitive operation swap; the insert ADT provides only primitive operations, like last that returns the last element of a sequence, etc. The AAM for this algorithm forms a tree of abstractions rooted at abst(insertion), shown in Figure 2. Various examples of abstractions and algorithm explanations are provided in Müldner (2003), Müldner and Shakshuki (2004) and Müldner *et al.* (2004, 2005).

Visualization. Associated visual representation may be used by the learner to help him or her understand the basic properties of this abstraction, such as invariants. It is possible to embed any web-viewable animations built by AA systems, such as Ganimal (Ganimal, 2007; Diehl *et al.*, 2002; Diehl and Kerren, 2002), Animal (Roßling and Freisleben, 2002), or JSamba (JSamba, 2007), as well as other formats, for example Marcomedia Flash (Macromedia, 2007) visualizations, animated GIFs, sound files, etc. As an example, Figure 3 shows a flash visualization of the insert() function of the insertion sort algorithm. Additional hyperlinks provide a description of fundamental concepts and an intuitive analogy, similar to HalVis.

Easy language transfer. SHALEX provides the intermediate representation of all AAM's primitive operations, called an abstract implementation model (AIM) (Figure 1). To implement the algorithm in a specific programming language, the learner has to

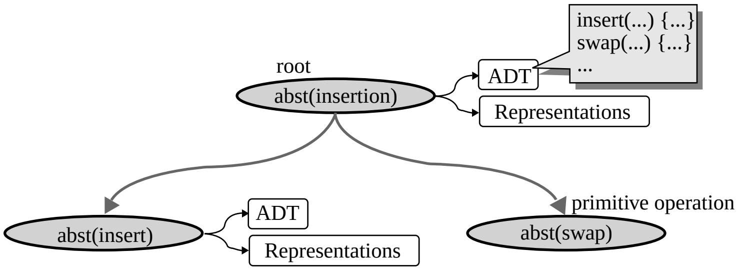


Figure 2.
An AAM for insertion sort

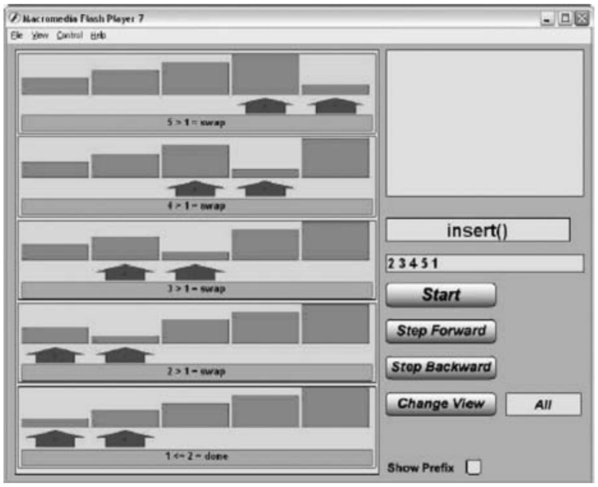


Figure 3.
Visualization of the insert of insertion sort

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.

Time complexity. 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, to our knowledge, is described in Pape and Schmitt (1997). The current version of SHALEX includes three kinds of tools designed to help the learner to derive the complexity of the algorithm being studied. In the first tool, based on Horstmann (2001), the learner can 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 and Tamassia (2001), provides visualization that helps to carry out time analysis of the algorithm. Finally, the third tool asks learners various questions regarding the time complexity of the algorithm being studied and evaluates their answers.

Learner and author models. SHALEX is an interactive system that allows the learner to select one of the available algorithms to study. It uses a learner model to record learner activities. These interactions are vital to support active learning (Hundhausen *et al.*, 2002). SHALEX helps the learner not only to understand what the algorithm is doing but also how the algorithm works; as well why the algorithm works (algorithm correctness).

In addition, it 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, and various AAM trees providing more or fewer abstractions. Authors' responsibilities include selecting tools to keep track of the learner performance. Instead of fixing a single tool such as asking a learner questions, SHALEX provides several tools including traditional tools, such as measuring the time spent on studying specific issues and comparing this time with author-specifies soft and hard deadlines, or keeping track of the percentage of questions that are correct answered by learner. More innovative tools supported by SHALEX include keeping track of user activities, such as selecting menu items, entering text fields, etc. The author then selects a specific tracking tool, and then decides on the adaptivity of the system. For example, the author model may also include assignments of various skill levels to the learner. If this is the case, then there will be two types of evaluation; to decide whether the learner's skill level should be changed, and to decide whether the learner has successfully learned the operation in question.

Additionally, our system has built-in features that help to evaluate the effectiveness of studying algorithms using this system. To compare the effectiveness of two different lessons for the same algorithm, the administrator may create two disjoint groups of students, and assign a different lesson to each group (a single algorithm may have one or more lessons, where two lessons may vary by the depth of their explanations, level of evaluation, etc.).

Authoring. The process of creating an algorithm explanation is supported by various tools, such as a library of existing lessons, and descriptions of ADTs. The author may fetch an existing item and adjust to her or his needs. A novel and essential feature of SHALEX is that it allows the author or the algorithm administrator to assign different modes of learning an algorithm: top-down, bottom-up and learnerselected.

In top-down learning, the learner studies the textual and optionally visual representation of the source node (i.e. the most abstract operation) of the AAM at first. Then, the learner studies all successor nodes and so on. The bottom-up learning approach is performed in an opposite direction, i.e. starting from leaves of the AAM. The learner-selected mode needs a more careful description. For any operation op that appears in the operation currently focused on, the learner may select op and request one of the following: help, taking a test (if the author decided to include testing), or explanation of this operation. In the first case, SHALEX provides a context-sensitive help. Specifically, based on the information available in the learner model, SHALEX provides a fundamental help (showing basic concepts), algorithm-specific help, or practice-oriented help (if the learner model indicates that the learner understood the algorithm but she or he had difficulties with problems that require manual simulation of this algorithm). In the second case, the learner may be given a test, and if the test is passed, the learner model will be updated. The author may specify that in order to complete studying the algorithm, the learner has to complete all tests, using evaluations available in the author model.

Note that explanations can also be built by the learners, who are permitted to play the role of authors. This way one can test and evaluate moderate constructivism ideas.

Implementation

SHALEX is being implemented in Java and XML. We briefly discuss the implementation of the most important technologies used in SHALEX: Java is used to implement the basic functionality and graphical user interfaces. With the help of XML, we represent system data (such as all algorithms, all users, nodes of AAM for specific algorithms, etc.) as well as the author and learner model. The information available to authors and/or learners can be rendered in a variety of ways, for example in HTML or PDF. XML data are made persistent using a native XML database, eXist (2007). When the learner requests the HTML view of all algorithms available in SHALEX, then the XML data are translated to HTML using XSLT (Tidwell, 2001) and displayed. The entire system has an open design, e.g. both models can be plugged into the system without changing the system's architecture. The current version of SHALEX is implemented as a client/server, multi-tier application. Users access the server with any web browser, and the server is implemented using servlets, which create dynamic web pages for the clients and communicate with the database tier. The implementation of SHALEX is an ongoing work.

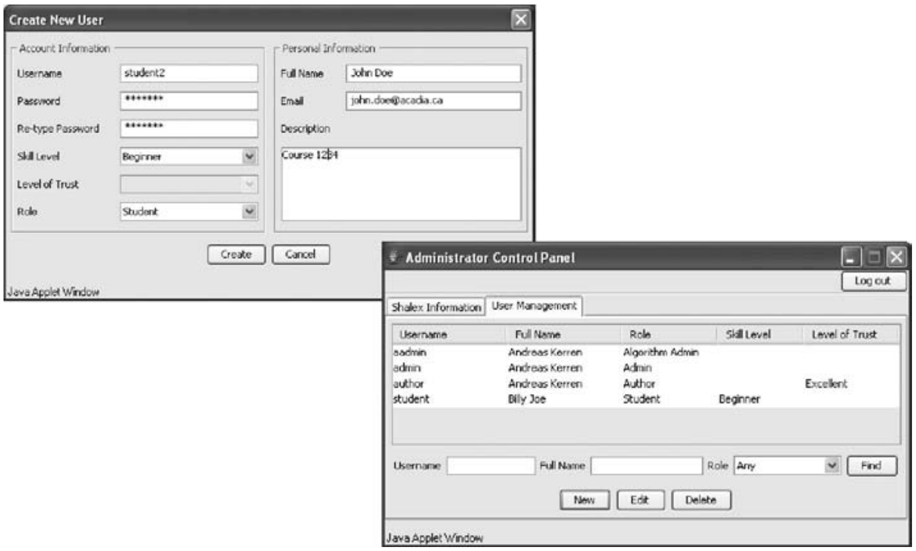
Case study: insertion sort

To exemplify the use of SHALEX on the basis of the different roles, we briefly highlight the most important steps in the development of algorithm explanations, user management, and the most important task: learning algorithms. As an example of an algorithm, we will use insertion sort again.

After the start of SHALEX, a login dialog appears. Here, the user has to authenticate and choose her/his assigned role, i.e. administrator, algorithm administrator, author, or student. We begin with the description of some technical aspects.

Administrating

Administrators are responsible for maintaining user accounts and the user's role, skill level (for students), or level of trust (for non-students) (Figure 4). These are purely



Username: student2
Password: *******
Re-type Password: *******
Skill Level: Beginner
Level of Trust:
Role: Student
Full Name: John Doe
Email: john.doe@acadia.ca
Description: Course 1234

Username	Full Name	Role	Skill Level	Level of Trust
aadmin	Andreas Kerren	Algorithm Admin		
admin	Andreas Kerren	Admin		
author	Andreas Kerren	Author		Excellent
student	Billy Joe	Student	Beginner	

Username: Full Name: Role: Any

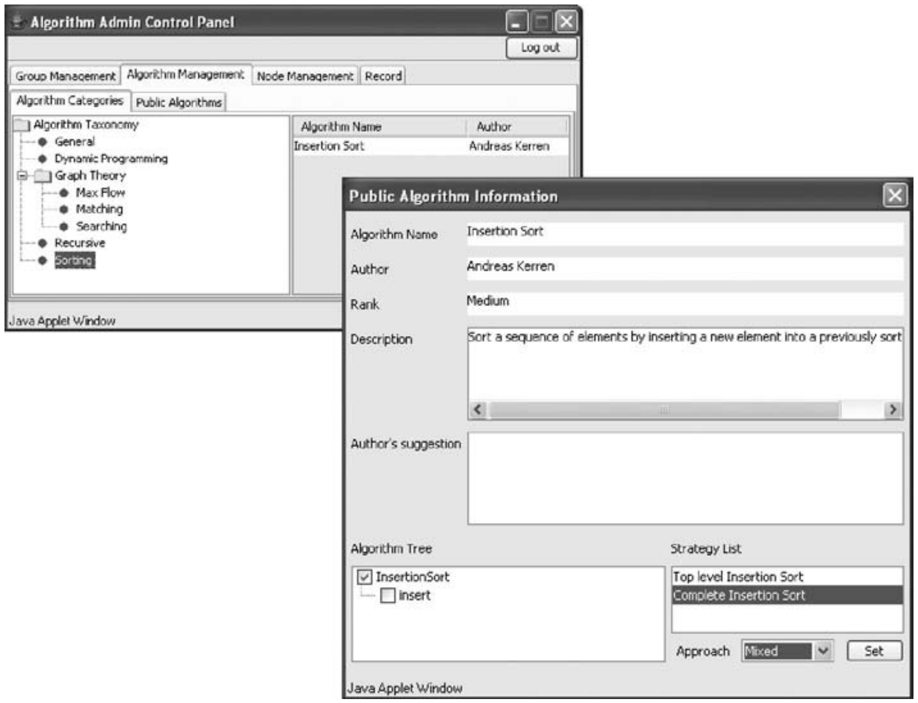
Figure 4.
Administrator Control Panel and creation of a new user

technical tasks and the administrator does not need to know anything about algorithms. Using the Administrator Control Panel, with a single mouse-click one can add a new user and additional information, such as e-mail address, length of study, or course number.

The responsibility of algorithm administrators is much broader. Within the Algorithm Admin Control Panel (Figure 5), they can define and manage student groups (controls are located on tab 1 of the upper screenshot example), e.g. if there are two parallel courses on algorithms or if one group should learn with visualizations using a top-down learning strategy and the other group should learn without visualizations using a bottom-up learning strategy. Furthermore, they can watch and record learner activities (tab 4) for evaluation purposes. Very important part of the algorithm administrator's activity is the management of all available algorithm explanations, i.e. different algorithms (tab 2, shown at the upper screenshot of Figure 5) as well as the nodes of the AAMs (tab 3): for example, the algorithm administrator can assign specific algorithms to student groups based on the algorithm taxonomy. Note that not all algorithms have to be publicly available. It is possible to hide some algorithms, e.g. for technical or didactical reasons. On the other hand, there is also taxonomy of all AAM nodes. Here, some basic concepts, such as string manipulation-related operations, can be found and published for common use of all authors. Selecting any published algorithm, SHALEX opens an information window about this algorithm in which all important learning procedures can be chosen including learning strategy (see below) and mode (top-down, bottom-up, or learner-selected (mixed)) (lower screenshot of Figure 5).

Authoring

For the preparation of an algorithm explanation, an author has to define an AAM for the algorithm to be explained. Let us use the AAM shown in Figure 2. To specify it in



Insertion Sort Andreas Kerren

Algorithm Name: Insertion Sort
Author: Andreas Kerren
Rank: Medium
Description: Sort a sequence of elements by inserting a new element into a previously sort

Algorithm Tree: ☑ InsertionSort; ☐ insert

Strategy List: Top level Insertion Sort; Complete Insertion Sort

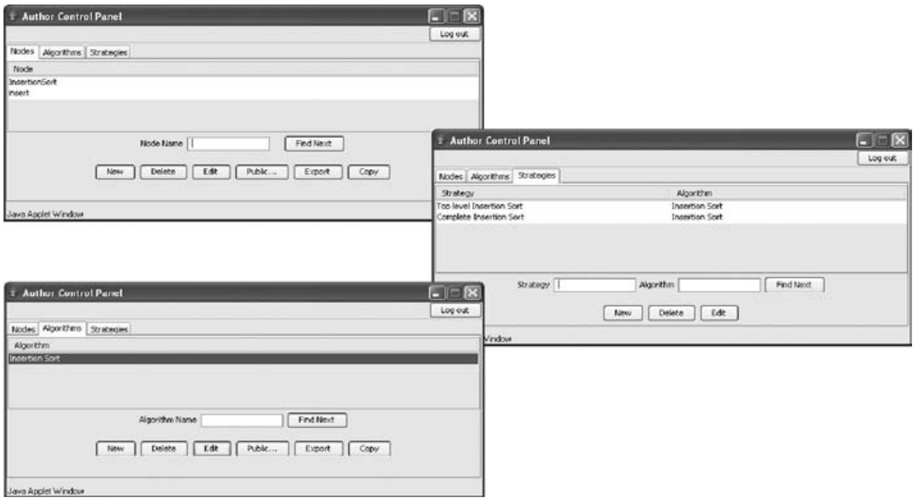
Approach: Mixed

Figure 5.
Algorithm Administrator Control Panel (only tab 2 is shown) with a pop-up window which contains information on a specific algorithm

SHALEX, the user must login as Author. In the Author Control Panel, he/she can define an AAM with the help of an easy to use point-and-click-interface (Figure 6).

We explain the functionality in a top-down manner: the author can insert/edit nodes of the AAM at the first tab of the Author Control Panel as well as algorithms (AAM trees) at the second tab. Nodes and trees can be specified individually and can be published once they are completed. It is also possible to reuse public nodes/algorithms or to copy public nodes/algorithms for own modifications. On the third tab, each algorithm can be annotated with a learning strategy which consists of none, one or several subtrees of the AAM. So, the author can individually control the granularity of explanations. In our screenshot example shown in Figure 6, two different strategies were defined: top-level learning of Insertion Sort and learning this algorithm by watching all nodes of the AAM, called "Complete Insertion Sort". For each algorithm, there is a hard time, specified by the strategy definition as the total time to study the algorithm that cannot be exceeded. Then, for each part (node) in this algorithm, there is a soft time; where the sum of all soft times is equal to the hard time. The learner is allowed to exceed a specific soft time, but then she/he would have to make up the lost time when studying other parts. This feature is very helpful for evaluation purposes.

All relevant information to edit/create a node can be entered into dialog boxes, as shown in Figure 7. Thus, pseudocode and informal descriptions together with an ADT are used to define an abstraction of an operation. The screenshot shows the definition of the insert node. This operation uses further four primitive operations:



Author Control Panel

Log out

Nodes | Algorithms | Strategies

Node
InsertionSort
insert

Node Name | Find Next

New | Delete | Edit | Public... | Export | Copy

Java Applet Window

Author Control Panel

Nodes | Algorithms | Strategies

Strategy	Algorithm
Top level Insertion Sort	Insertion Sort
Complete Insertion Sort	Insertion Sort

Strategy | Algorithm | Find Next

New | Delete | Edit

Author Control Panel

Nodes | Algorithms | Strategies

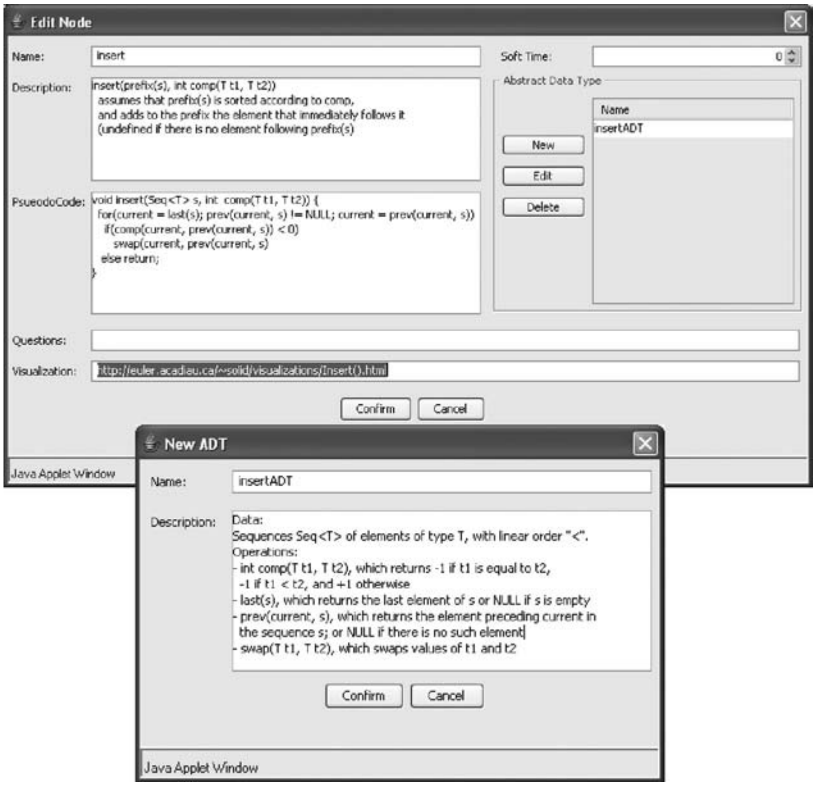
Algorithm
Insertion Sort

Algorithm Name | Find Next

New | Delete | Edit | Public... | Export | Copy

Java Applet Window

Figure 6.
The three tabs of the Author Control Panel



Edit Node

Name: insert

Soft Time: 0

Description:

```
insert(prefix(s), int comp(T t1, T t2))
  assumes that prefix(s) is sorted according to comp,
  and adds to the prefix the element that immediately follows it
  (undefined if there is no element following prefix(s)
```

PsueodoCode:

```
void insert(Seq<T> s, int comp(T t1, T t2)) {
  for(current = last(s); prev(current, s) != NULL; current = prev(current, s))
    if(comp(current, prev(current, s)) < 0)
      swap(current, prev(current, s)
    else return;
}
```

Abstract Data Type

Name
insertADT

New | Edit | Delete

Questions:

Visualization: http://euler.acadiau.ca/~solid/visualizations/Insert().html

Confirm | Cancel

Java Applet Window

New ADT

Name: insertADT

Description:

Data:
Sequences Seq<T> of elements of type T, with linear order "<".
Operations:
- int comp(T t1, T t2), which returns -1 if t1 is equal to t2,
  -1 if t1 < t2, and +1 otherwise
- last(s), which returns the last element of s or NULL if s is empty
- prev(current, s), which returns the element preceding current in
  the sequence s; or NULL if there is no such element
- swap(T t1, T t2), which swaps values of t1 and t2

Confirm | Cancel

Java Applet Window

Figure 7.
Edit a node and new ADT

(1) comp(T t1, T t2) – compares two elements of type T;
(2) last(s) – returns the last element of a sequence s;
(3) prev(current, s) – returns the element of the sequence s, preceding current; and
(4) swap(T t1, T t2) – swaps t1 and t2 of type T.

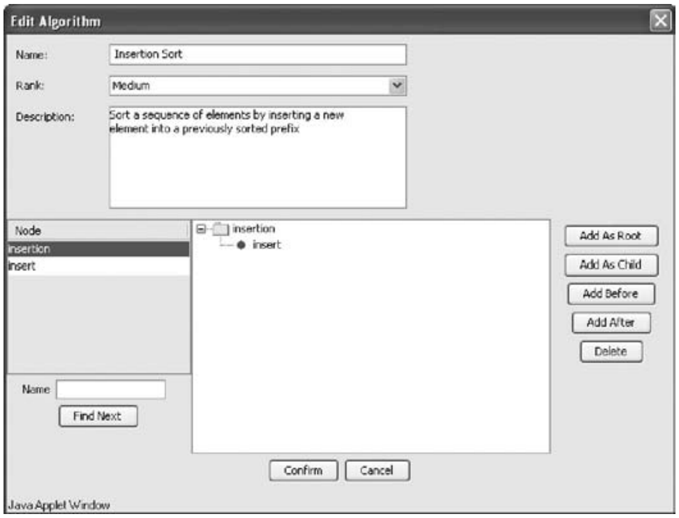
They can be declared within the insert ADT (Figure 7) together with required data structures: in our case, sequences Seq < T > of ordered elements of type T. Based on this information, a possible pseudocode implementation of the insert operation is given below:

```
void insert(Seq<T>s, int comp(T t1, T t2)) {
   for(current=last(s); prev(current, s) ! = NULL; current = prev(current, s))
      if(comp(current, prev(current, s))<0)
         swap(current, prev(current, s))
      else return;
}
```

Optionally, the author can indicate an appropriate visualization or the source of an interactive questionnaire that will be displayed in separate windows if the learner studies this node at learning time. Figure 3 shows a simple example visualization of the node of the insert operation.

After the specifications of individual nodes, the author can easily build up the entire AAM tree by a self-explanatory and comfortable point-and-click-interface shown in Figure 8. In our running example, we have only two nodes: insertion that will be the root of the tree and insert that will be its only child which is not primitive. Remember that the swap operation used by insert was declared as primitive operation.

If the AAM is ready, SHALEX supports the definition of one or more learning strategies for an algorithm. The correspondent dialog box is shown in Figure 9. In the center of the dialog box, the entire AAM tree is displayed using a standard explorer layout for trees. Thus, single nodes or entire subtrees can be marked for consideration

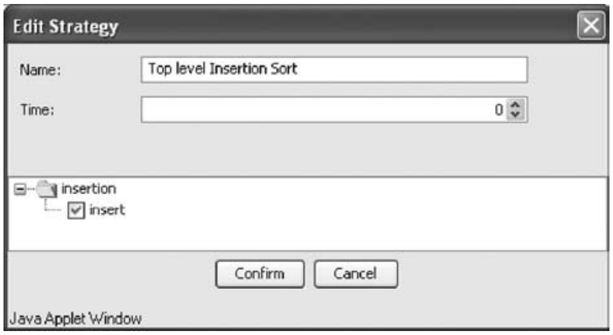

Edit Algorithm

Name: Insertion Sort
Rank: Medium
Description: Sort a sequence of elements by inserting a new element into a previously sorted prefix

Node
insertion
insert

insertion
  insert

Figure 8.
Dialog box for edit algorithms (AAMs)



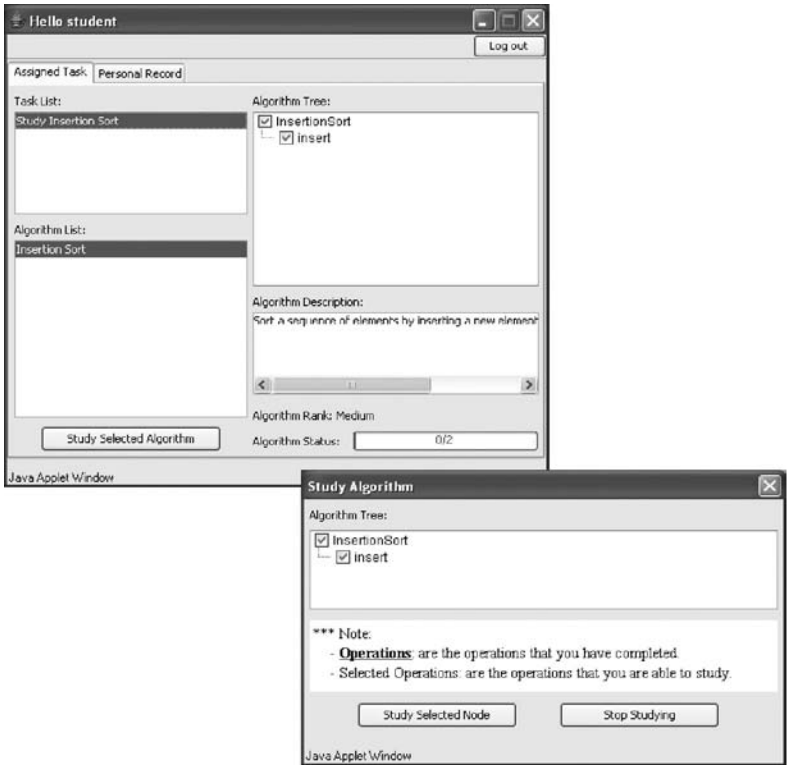
Name: Top level Insertion Sort
Time: 0

Figure 9.
Define a learning strategy

in the learning process of the students. It is also possible to choose a hard time for studying the algorithm as described before.

Learning

After login as Student, SHALEX offers a learning panel with assigned tasks, as shown in Figure 10. All tasks are itemized in a task list. For example, if the student attends a course on algorithms then the task list could contain several learning tasks, such as "Study Sorting Algorithms" or "Study Geometric Algorithms". Thus, each task can



Sort a sequence of elements by inserting a new element

Algorithm Rank: Medium

Algorithm Status: 0/2

*** Note:
- **Operations**: are the operations that you have completed.
- Selected Operations: are the operations that you are able to study.

Figure 10.
Student environment

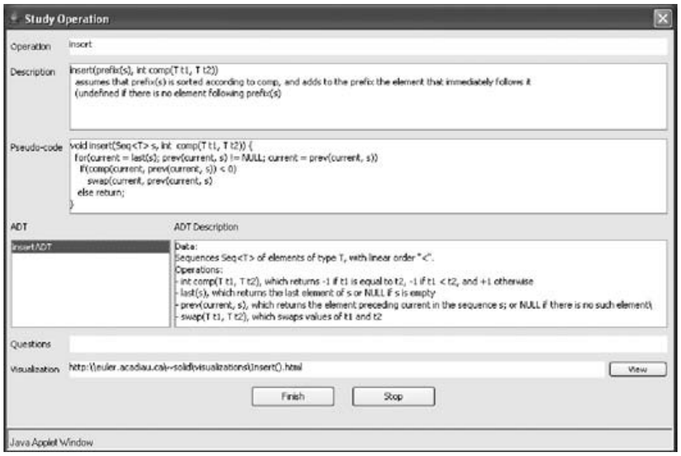
imply one or several related algorithms. In our running example, we only have the insertion sort algorithm. Choosing this example, the AAM tree and a brief description of the selected algorithm is displayed on the right-hand side of the learning panel. Assume the author has chosen a learning strategy for this student's group that allows free learning and watching of all AAM nodes of the selected algorithm (complete insertion sort). In our example, this policy is symbolized by check marks at all nodes. Furthermore, the algorithm was ranked as "Medium" by the author of this algorithm explanation. To reflect the current status of the algorithm, a status bar shows how many nodes of the AAM have been successfully completed.

If the student decides to study a specific algorithm, a new window with the current AAM appears, as shown in Figure 10. Here, the student can select any node of the displayed AAM tree (recall that we are using "free view" rather than a more restrictive strategy such as "top down") and click on the "Study Selected Node" button for learning. Let us assume that the insert node has been chosen. Then, a new panel appears, as shown in Figure 11.

This "Study Operation" panel contains all explanations and information provided by the author for the selected node. All fields of this panel can contain hyperlinks to both external and internal sources. In this way, hypermedia can be effectively used to explain single primitive operations or to substantiate details in the algorithm description. Visualization and a questionnaire can be invoked by the student too, as described before and shown in Figure 3.

Conclusions and future work

This paper presented our proposed system for explaining algorithms, which is based on structured hypermedia approach. It has been shown that the 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 learner model to provide spatial and temporal links; and the use of XML to store information. We summarize our contributions in more detail in the following subsection.



Operation: insert

Description:
```
insert(prefix(s), int comp(T t1, T t2))
  assumes that prefix(s) is sorted according to comp, and adds to the prefix the element that immediately follows it
  (undefined if there is no element following prefix(s)
```

Pseudo-code:
```
void insert(Seq<T> s, int  comp(T t1, T t2)) {
  for(current = last(s); prev(current, s) != NULL; current = prev(current, s))
    if(comp(current, prev(current, s)) < 0)
      swap(current, prev(current, s)
   else return;
}
```

ADT: insertADT

ADT Description:
```
Data:
Sequences Seq<T> of elements of type T, with linear order "<".
Operations:
- int comp(T t1, T t2), which returns -1 if t1 is equal to t2, -1 if t1 < t2, and +1 otherwise
- last(s), which returns the last element of s or NULL if s is empty
- prev(current, s), which returns the element preceding current in the sequence s; or NULL if there is no such element\
- swap(T t1, T t2), which swaps values of t1 and t2
```

Questions:

Visualization: http:\\euler.acadiau.ca\~solid\visualizations\Insert().html

Figure 11.
Learning a specific operation

Contributions

In this paper, we presented a novel algorithm explanation system, whose most important features include:

(1) Active learning; students can:
   - enter their own inputs; a do-it-yourself mode and predict the next step of the algorithm, or determine the essential algorithm properties;
   - develop their own algorithm explanations rather than use the existing explanations prepared by experts; and
   - use the pseudo-code available in the algorithm explanation to implement this algorithm in a selected programming language.

(2) Multiple levels of abstraction; each level has its own pseudo-code textual representation, visual representation which exposes its properties (particular its invariants), and questions that help the learner to perform self-evaluation.

(3) Internal graph representation of the explanation, which is transparent to the user and makes it possible to learn the algorithm using one of three existing strategies: a top-down approach, a bottom-up approach, or a mix of both.

(4) Support for hypermedia, which is not limited to graphics and animations, but also includes internal hyperlinks (pointing to other parts of the related explanations) and external links to other web sites (such as web sites that provide relevant definitions).

(5) Internal messaging system that can be used to exchange information between various users.

(6) Four different roles that can be played by users; including learners, authors (responsible for creating algorithm explanations and various lessons), algorithm administrators (responsible for management of groups of users by assigning them to study specific algorithms, etc.), and finally system administrators (responsible for maintaining user accounts, their roles and maintaining groups of users).

(7) Persistent storage that not only stores algorithm explanations but also the student model, which can be analyzed to provide feedback, prepare reports showing the learners performance, and provide system adaptations (such as various kinds of help).

(8) Flexible implementation of the student model, which makes it possible to store in the model various kinds of information, such as marks for answered questions or learners interactions with the system.

Future work

The educational benefit of our approach has to be proven by accurate evaluation. The good results of several evaluations of the related CK approach (Baloian *et al.*, 2005) support our assumption that an empirical evaluation of SHALEX will yield to good results.

The first versions of algorithm visualizations were implemented using Macromedia (2007) Flash. For the next version, we are considering using HTML pages to display more complex and interactive visualizations (this follows the design of Ganimal

(Diehl and Kerren, 2002)). Additionally, some improvements related the GUI will be implemented, for example, the AAM of an algorithm should be displayed as a real graph in the GUI and not as a tree.

After using the results of the usability check to improve SHALEX and fix possible bugs, we will design a quantitative evaluation with exercises for a performance test to compare SHALEX with other AA systems and/or the CK approach. In order to make our system usable, we are also planning on performing evaluations in class with students from computer science at various universities.

References

Aho, A.V., Hopcroft, J.E. and Ullman, J.D. (1983), *Data Structures and Algorithms*, Addison-Wesley, Reading, MA.

Baloian, N., Middleton, C., Breuer, H. and Luther, W. (2005), "Algorithm visualization using concept keyboards", *Proceedings of the ACM Symposium on Software Visualization (SoftVis '05)*, ACM, St Louis, MO, pp. 7-16.

Baumgartner, P. (1999), "Evaluation of media-based learning (in German)", in Kindt, M. (Ed.), *Projektevaluation in der Lehre – Multimedia an Hochschulen zeigt Profil(e)*, Waxmann, Münster, pp. 61-97.

Biermann, H. and Cole, R. (1999), *Comic Strips for Algorithm Visualization*, Tech. Rep. 1999-778, NYU, New York, NY.

Braune, B. and Wilhelm, R. (2000), "Focusing in algorithm animation", *IEEE Transactions on Visualization and Computer Graphics*, Vol. 6 No. 1, pp. 1-7.

Brown, M.H. (1988), "Perspectives on algorithm animation", *Proceedings of the ACM SIGCHI '88 Conference on Human Factors in Computing Systems, May*, ACM, Washington, DC, pp. 33-8.

Cox, K.C. and Roman, G-C. (1992), "Abstraction in algorithm animation", *Proceedings of the 1992 IEEE Workshop on Visual Languages, IEEE, September*, IEEE Computer Society Press, Seattle, WA, pp. 18-24.

Diehl, S. (Ed.) (2002), "Software visualization", *Vol. 2269 of LNCS State-of-the-Art Survey*, Springer, Berlin.

Diehl, S. and Kerren, A. (2002), "Reification of program points for visual execution", *Proceedings of the First IEEE International Workshop on Visualizing Software for Understanding and Analysis (VisSoft '02). IEEE, Jun*, IEEE Computing Society Press, Paris, pp. 100-9.

Diehl, S., Görg, C. and Kerren, A. (2002), "Animating algorithms live and post mortem", in Diehl, S. (Ed.), *Software Visualization. Vol. 2269 of LNCS Stateof-the-Art Survey*, Springer, Berlin, pp. 46-57.

Eades, P. and Zhang, K. (Eds) (1996), *Software Visualization*, World Scientific Publisher, Singapore.

eXist (2007), "Open source native XML database", available at: http://exist.sourceforge.net/

Fleischer, R. and Kucera, L. (2002), "Algorithm animation for teaching", in Diehl, S. (Ed.), *Software Visualization. Vol. 2269 of LNCS State-of-the-Art Survey*, Springer, Berlin, pp. 113-28.

Ganimal (2007), "Project homepage", available at: www.cs.uni-sb.de/GANIMAL

Gloor, P.A. (1992), "AACE – algorithm animation for computer science education", *Proceedings of the 1992 IEEEWorkshop on Visual Languages, September*, IEEE, Seattle, WA, pp. 25-31.

Gloor, P.A. (1998a), "Animated algorithms", in Stasko, J., Domingue, J., Brown, M.H. and Price, B.A. (Eds), *Software Visualization: Programming as a Multimedia Experience*, Chapter 27, MIT Press, Cambridge, MA, pp. 409-16.

Gloor, P.A. (1998b), "User interface issues for algorithm animation", in Stasko, J., Domingue, J., Brown, M.H. and Price, B.A. (Eds), *Software Visualization: Programming as a Multimedia Experience*, Chapter 11, MIT Press, Cambridge, MA, pp. 145-52.

Goodrich, M. and Tamassia, R. (2001), *Data Structures and Algorithms in Java*, 2nd ed., Wiley, New York, NY.

Hansen, S.R., Narayanan, N.H. and Hegarty, M. (2002), "Designing educationally effective algorithm visualizations: embedding analogies and animations in hypermedia", *Journal of Visual Languages and Computing*, Vol. 13 No. 3, pp. 291-317.

Horstmann, C. (2001), *Big Java: Programming and Practice*, Wiley, New York, NY.

Hundhausen, C., Douglas, S. and Stasko, J. (2002), "A meta-study of algorithm visualization effectiveness", *Journal of Visual Languages and Computing*, Vol. 13 No. 3, pp. 259-90.

Johannes, D., Seidel, R. and Wilhelm, R. (2005), "Algorithm animation using shape analysis: visualising abstract executions", *Proceedings of the ACM Symposium on Software Visualization (SoftVis '05)*, ACM, St Louis, MO, pp. 17-26.

JSamba (2007), "Project homepage", available at: www-static.cc.gatech.edu/gvu/softviz/algoanim/jsamba/

Kerren, A. (2004a), "Generation as method for explorative learning in computer science education", *Proceedings of the 9th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '04), ACM*, ACM Press, Leeds, pp. 77-81.

Kerren, A. (2004b), "Learning by generation in computer science education", *Journal of Computer Science & Technology (JCS&T)*, Vol. 4 No. 2, pp. 84-90.

Kerren, A. and Stasko, J.T. (2002), "Algorithm animation – chapter introduction", in Diehl, S. (Ed.), *Software Visualization, Vol. 2269 of LNCS State-of-the-Art Survey*, Springer, Berlin, pp. 1-15.

Macromedia (2007), "Flash", available at: www.macromedia.com/software/flash/

Müldner, T. (2003), "An algorithm for explaining algorithms", Tech. Rep. TR-2003-01, Jodrey School of Computer Science, Acadia University, available at: http://cs.acadiau.ca/technicalReports/

Müldner, T. and Shakshuki, E. (2004), "On visualization and implementation of algorithms", *Proceedings of the 5th International Conference on Information Technology Based Higher Education & Training (ITHET '04), IEEE*, IEEE Computer Society Press, Istanbul, pp. 138-43.

Müldner, T., Shakshuki, E. and Merill, J. (2004), "Selecting media for explaining algorithms", *Proceedings of the AACE World Conference on Educational Multimedia, Hypermedia and Telecommunications (EDMEDIA '04)*, AACE, Lugano, Swizerland, pp. 2048-53.

Müldner, T., Shakshuki, E., Kerren, A., Shen, Z. and Bai, X. (2005), "Using structured hypermedia to explain algorithms", *Proceedings of the 3rd IADIS International Conference e-Society '05*, IADIS, Qawra, pp. 499-503.

Naps, T.L. (2005), "JAV'E: supporting algorithm animation", *IEEE Computer Graphics and Applications*, Vol. 25 No. 5, pp. 49-55.

Pape, C. and Schmitt, P.H. (1997), "Visualizations for proof presentation in theoretical computer science education", in Halim, Z., Ottmann, T. and Razak, Z. (Eds), *Proceedings of International Conference on Computers in Education (ICCE '97), AACE – Association for the Advancement of Computing in Education*, pp. 229-36.

Petre, M., Baecker, R. and Small, I. (1998a), "An introduction to software visualization", in Stasko, J.T., Domingue, J., Brown, M.H. and Price, B.A. (Eds), *Software Visualization*, MIT Press, Cambridge, MA, pp. 3-26.

Petre, M., Blackwell, A.F. and Green, T.R.G. (1998b), "Cognitive questions in software visualization", in Stasko, J.T., Domingue, J., Brown, M.H. and Price, B.A. (Eds), *Software Visualization*, MIT Press, Cambridge, MA, pp. 453-80.

Price, B.A., Baecker, R. and Small, I. (1993), "A principled taxonomy of software visualization", *Journal of Visual Languages and Computing*, Vol. 4 No. 3, pp. 211-66.

Roßling, G. and Freisleben, B. (2002), "ANIMAL: a system for supporting multiple roles in algorithm animation", *Journal of Visual Languages and Computing*, Vol. 13 No. 3, pp. 341-54.

SHALEX (2007), "Project homepage", available at: http://cs.acadiau.ca~/solid/ae.htm

Stasko, J.T., Domingue, J., Brown, M.H. and Price, B.A. (1998), *Software Visualization*, MIT Press, Cambridge, MA.

Tidwell, D. (2001), *XSLT*, O'Reilly, Sebastopol.

Wilhelm, R., Müldner, T. and Seidel, R. (2002), "Algorithm explanation: visualizing abstract states and invariants", in Diehl, S. (Ed.), *Software Visualization, Vol. 2269 of LNCS State-of-the-Art Survey*, Springer, Berlin, pp. 381-94.

Further reading

Kerren, A., Müldner, T. and Shakshuki, E. (2006), "Novel algorithm explanation techniques for improving algorithm teaching", *Proceedings of the 3rd ACM Symposium on Software Visualization (SoftVis '06)*, ACM Press, Brighton, pp. 175-6.

Roßling, G., Naps, T., Hall, M., Karavirta, V., Kerren, A., Leska, C., Moreno, A., Oechsle, R., Rodger, S.H., Urquiza-Fuentes, J. and Velźquez-Iturbide, J.A. (2006), "Merging interactive visualizations with hypertextbooks and course management", *ACM SIGCSE Bulletin – Inroads*, Vol. 38 No. 4, pp. 166-81.

Corresponding author
Elhadi Shakshuki can be contacted at: elhadi.shakshuki@acadiau.ca