Algorithm Explanation Using Multimodal Interfaces

Nelson Baloian
Universidad de Chile
nbaloian@dcc.uchile.cl

Abstract

Software visualization and algorithm animation have been tackled almost exclusively from the visual point of view: this means representation and control occur through the visual channel. This approach has its limitations. To achieve better comprehension, we deal with multimodal interfaces that include other means of interaction together with those of the standard systems for data visualization and algorithm animation. The notion of specific concept keyboards is introduced. As a consequence, modern information and learning systems for algorithm animation are enhanced in such a way that control and interaction take place through appropriate interfaces designed for the special purpose so that not only sighted but also blind users can navigate within these systems. We provide some examples to show the relevance of this new approach.

Categories and subject descriptors

H.5.2 [Information Interfaces and Presentation (e.g., HCI)]: User Interfaces, Input devices and strategies, interaction styles; K.3.1 [Computers and Education]: Computer Uses in Education

Keywords

Concept keyboards, algorithm animation, multimodal interfaces, disabled persons

1. Introduction

There is a growing interest in algorithm animation (AA) to help instructors explain algorithms and learners understand them. A certain algorithm animation shows relevant parameters and variables, the current state, a suitable representation of the objects being manipulated and often an animated formal description of the algorithm.

Complex model structures are simplified to highlight the important aspects by omitting non-relevant details. To achieve better comprehension, the designer scales down data to coarser structures and slows down the speed of algorithms that process data. Smooth transitions between different states of moving objects, for example, elements inside an array while performing a sorting algorithm, can help to follow the way it works on graphic representations of data structures. In most cases algorithms work on data structures only locally in a serial way. So learners can run the algorithm step-by-step selecting different input parameters, step size and execution speed while following a configurable animation of algorithms. The purpose of this kind of interactive software is to gain insight into the dynamic behavior of the algorithm [27].

It has been noted that there have been only minor attempts to produce input interfaces mirroring the concepts and data structures of the algorithms. In most cases, user interaction and output visualization are not separate. The system only offers a simple toolbar that allows the user to reset the algorithm and go forwards and backwards; sometimes redo and undo functionality is implemented, but no context-dependent navigation facilities are provided. Consequently, it seems promising to replace the standard control using mouse clicks on graphic displays with alternative ways to generate concept keyboards (CKs) mirroring the inherent logical structures of the proposed algorithm. Concept Keyboards are keyboards with a reduced set of keys, but each one will trigger a more complex action in the context of the task being accomplished by the system. For example, the pressing of a single key may cause a sorting algorithm animation to exchange the positions of two elements in the array. Experience with CKs will help a student to understand how an algorithm works by allowing him to become involved in the execution of the algorithm and to navigate through the data structure. The user's attempts to perform actions on algorithms and data structures are reflected by changes in the visualization or another form of output like textual or acoustic information. This enables us to provide
users with interfaces that enhance the comprehension of the algorithm being presented and that are even suitable for people with sensory disabilities, since the output information uses not only the visual but also the acoustic channel.

Our methodology is summarized in the following concepts, which can be used to create other interaction and reception modes in parallel to classic control through buttons and menus [5], [6]:

- Define a level of detail the user is supposed to learn about the algorithm and choose a corresponding implementation of it. This means defining the operations the learner should be aware of and implementing them as a single function or method.
- Develop a suitable CK for the algorithm implementation by redefining keys or creating special button schemes on a graphic tablet. Then the keys represent the basic operations performed by the algorithm.
- Develop a spatial arrangement of the keyboard that mirrors the calling hierarchy of the procedures or methods of the algorithm. Implement a tool that delegates this task to the learner.
- Use icons or earcons to enhance the functionality of a specific key area design.
- Let the user choose which step should be executed at any stage of executing the algorithm.
- Define appropriate output channels combining a visualization with natural speaking or text.
- Implement an efficiency control which gives users optional feedback about the quality of their solutions.

2. Concept keyboards

Concept keyboards are used in different contexts. Often, compared with a normal keyboard, keys are broader and their number is reduced. They are used to trigger more complex semantic actions over the system in which they were implemented. Special software supplied allows one to redefine the function of each key and to regroup keys into fields of differing size.

Another way to implement concept keyboards is by programming touch screens or tablets. It is possible to associate them to individually shaped domains or icons with textual or graphic content. Their functionality also includes the launching of sound files or programs. Pens or fingers press the keys, and a prerecorded sound can be played or a picture displayed when they are pressed. Alternatively, the fields can also be accessed with push buttons or with joysticks. Objects can be placed over sensitive areas of the concept keyboard with different overlays. These can be produced by a kind of drawing program or selected from a large picture library. The IntelliKeys overlay keyboard [25] is a commercial product which provides physical, visual and cognitive access for people with a wide range of disabilities.

The concept keyboard is considerably simpler to manipulate for children or impaired people. Since keys are bigger, they are suitable for people with residual sight. For totally blind people tactile exercise sheets (called overlays) are installed on the concept keyboard. Other applications are intended for students with learning difficulties. Our working hypothesis first introduced in Baloian and Luther [2], [3] will be that not only sighted but also impaired users should navigate within our systems through new or augmented interfaces, using enhanced perception tools to achieve algorithm animation and visualization. To support multiple types of users with multiple types of interfaces, it is necessary to achieve the paradigm of a strict separation of view and control. Thus, it seems worthwhile to think about adequate concept keyboards supporting direct navigation on internal logical structures. This approach helps the user rebuild the original structures in mind as far as possible. In any case, it is well known that blind people develop special forms of navigating within an unknown environment and represent spatial structure with cognitive difficulty [4]. This is the case in the real world as well as in virtual computer-based environments. Evidently, sighted users are not excluded; on the contrary, we claim that complementary perception channels and navigating facilities will represent a real enhancement for “normal users”.

To achieve this goal, we create appropriate key-boards consisting of well-arranged key areas combined with expressive icons and earcons that allow direct interaction with model objects. An earcon characterized by a representative melody can be added to any iconic control object. Earcons are abstract musical tones that can be used in structured combinations to create auditory messages. In 1989 Meera Blattner [7] introduced earcons as nonverbal audio messages to provide information to the user about computer objects, operations or interactions. Earcons are constructed from simple, easily recognizable melodies also called motifs. Interfaces augmented by these complementary perception channels do indeed provide better comprehension.

3. Controller design

To implement the methodology presented in the first section, we designed an architecture providing a number of components and an inspector functionality.
The system was implemented in Java, so we will describe it by their classes. The first component is called algorithm implementation (AGI). This corresponds to the principal class implementing the algorithm, which is loaded on the fly by reading its classname and location. The Reflection API allows a Java application to acquire the definitions of classes and operate on them at runtime without having them available at compile time. The library supports the instantiation of a class from its text name, acquiring information on a class and calling methods on an object. Once an instance of a class has been created, the controller can access meaningful methods that execute the algorithm action-by-action with a certain granularity for manipulating the data structure. These methods are described in the AGIXML (see Figure 1) as are the data structure related to the algorithm, an array of numbers (to be sorted), and a pointer defining the focus on a node of the tree or an integer value as the total number of nodes.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE dataStructure SYSTEM "dataStructure.dtd">
<dataStructure>
  <className>HeapSort</className>
  <packageName> </packageName>
  <data id="0">
    <source sourceType="Variable" forbidUserInput="YES">
      <sourceName>array</sourceName>
    </source>
    <objectType>java.util.ArrayList</objectType>
  </data>
  <data id="1">
    <source sourceType="Variable" forbidUserInput="YES">
      <sourceName>indexFocus</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </data>
  <data id="2">
    <source sourceType="Variable" forbidUserInput="YES">
      <sourceName>numElements</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </data>
</dataStructure>
```

**Figure 1: AGIXML file concerning the HeapSort algorithm**

The IOXML file (see Figure 2) specifies the concept keyboard or alternative input devices used, a data input frame and one or several output devices. By pressing a key on the CK, the user selects a useful operation (Action Selection). The controller then asks him for the necessary parameters, which are introduced through the input frame (Parameter Input). Then the action is executed, leading to a modified data structure. The controller instructs the output device to represent the updated data structure (Output). Thus, the principal task of the controller is to start and to synchronize the four components AGI, AS, PI, and O. All information relevant to the task is found in the XML files. The task can be launched by the user with the command `Java Principal Classname.xml io_Classname.xml`.

The Heapsort AGI includes thirteen methods, allowing building or deleting the tree (insert, append, deleteTree), navigating within the tree (moveUp, goLeft, goRight, setFocus), establishing the heap property in a subtree (findBiggerChild, swapBiggerChild, percolateElementdown), removing and replacing the root with the last leaf (removeRoot) and testing the heap or sub-heap property. Notice that only the controller is aware of the current collection of methods. It activates the actions regardless of whether or not the entire AGI is capable of executing the algorithm.

```xml
<?xml version="1.0"?>
<DOCTYPE ioDevices SYSTEM "ioDevices.dtd">
<ioDevices>
  <device io="SELECT">
    <className>KeyboardFrame</className>
    <xmlFile>kbd_HeapSort.xml</xmlFile>
  </device>
  <device io="ARGUMENT">
    <className>InputFrame</className>
  </device>
  <device io="OUTPUT">
    <className>BinaryTree</className>
    <xmlFile>BinaryTree.xml</xmlFile>
    <xmlJoin>join_BinaryTreeHeapSort.xml</xmlJoin>
  </device>
  <device io="OUTPUT">
    <className>BinaryTreeSpeech</className>
    <xmlFile>BinaryTreeSpeech.xml</xmlFile>
    <xmlJoin>join_BinaryTreeSpeechHeapSort.xml</xmlJoin>
  </device>
</ioDevices>
```

```xml
<action id="13">
  <description>deleteTree</description>
  <methodName>deleteTree</methodName>
  <help> deleteTree erases the actual tree in order to build a new one. </help>
</action>
```

```xml
<action id="0">
  <description>Insert</description>
  <methodName>insert</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>element</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> Insert a new node into the tree. If the tree has the heap property, the new node is inserted followed by heapifying the tree. Otherwise the node is appended at the bottom of the tree. </help>
</action>
```

```xml
<action id="11">
  <description>moveUp</description>
  <methodName>moveUp</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>focus</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> moveUp moves the focus up in the tree. </help>
</action>
```

```xml
<action id="12">
  <description>goLeft</description>
  <methodName>goLeft</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>focus</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> goLeft moves the focus left in the tree. </help>
</action>
```

```xml
<action id="13">
  <description>goRight</description>
  <methodName>goRight</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>focus</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> goRight moves the focus right in the tree. </help>
</action>
```

```xml
<action id="14">
  <description>setFocus</description>
  <methodName>setFocus</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>focus</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> setFocus selects a node in the tree. </help>
</action>
```

```xml
<action id="15">
  <description>findBiggerChild</description>
  <methodName>findBiggerChild</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>focus</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> findBiggerChild finds the child node with the larger value. </help>
</action>
```

```xml
<action id="16">
  <description>swapBiggerChild</description>
  <methodName>swapBiggerChild</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>focus</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> swapBiggerChild swaps the child node with the larger value. </help>
</action>
```

```xml
<action id="17">
  <description>percolateElementdown</description>
  <methodName>percolateElementdown</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>focus</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> percolateElementdown percolates the element down in the tree. </help>
</action>
```

```xml
<action id="18">
  <description>removeRoot</description>
  <methodName>removeRoot</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>focus</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> removeRoot removes the root node of the tree. </help>
</action>
```

```xml
<action id="19">
  <description>addLeaf</description>
  <methodName>addLeaf</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>leaf</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> addLeaf adds a new leaf node to the tree. </help>
</action>
```

```xml
<action id="20">
  <description>addLeafWithFocus</description>
  <methodName>addLeafWithFocus</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>focus</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> addLeafWithFocus adds a new leaf node to the tree with a specific focus. </help>
</action>
```

```xml
<action id="21">
  <description>removeLeaf</description>
  <methodName>removeLeaf</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>focus</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> removeLeaf removes a leaf node from the tree. </help>
</action>
```

```xml
<action id="22">
  <description>addFocus</description>
  <methodName>addFocus</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>focus</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> addFocus adds a new focus node to the tree. </help>
</action>
```

```xml
<action id="23">
  <description>removeFocus</description>
  <methodName>removeFocus</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>focus</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> removeFocus removes a focus node from the tree. </help>
</action>
```

```xml
<action id="24">
  <description>setFocusToLeaf</description>
  <methodName>setFocusToLeaf</methodName>
  <methodArgument>
    <source sourceType="UserInput" forbidUserInput="NO">
      <sourceName>focus</sourceName>
    </source>
    <objectType>java.lang.Integer</objectType>
  </methodArgument>
  <help> setFocusToLeaf sets the focus to a leaf node. </help>
</action>
```
In this example the IOXML performs a configuration task. It selects a virtual concept keyboard, which is defined in `kbd_HeapSort.xml`, sets up an input frame to collect the parameter input and establishes a graphical output through the `BinaryTree.xml` and a synthesized speech output by evaluating the `BinaryTreeSpeech.xml` file. The `join_XML` file links Output- and AGI-objects and their content; the current focus is linked with the node in red. The keyboard is represented by a 4/4 grid. Each position in the grid is named after the column and row to which it belongs. With the XML file defined, the designer determines the position of each of the action keys on the CK. The keys in the keyboard can be customized by modifying the label and the optional sound associated with them to display the name of the action or additional helpful hints.

The algorithm controller is highly configurable and the relevant input/output classes are determined at runtime. Once instances of the classes have been created, these objects can be used to acquire reflective information, to acquire and set values and to invoke methods on objects. The action list and the configuration file for the input devices are communicated. An important public class is the IOManager, which handles the IODevices and instantiates the ClipManager. During input and output activities several acoustic messages can be generated. Thus, the ClipManager administrates the sound files by way of the methods `playClip` or `readString`, which reads out a string letter-wise. To synchronize the vocal messages, a FIFO-queue is created, and the clips are preferred, depending on their relevance. If a more important clip should be displayed, the current clip is interrupted.

The first version of our software implements the acoustic output via prefabricated wav-files. Actually, we aim to integrate a text-to-speech software that generates the words on the fly. Up to now, the algorithm animation generates only discrete snapshots of the data structure, and no smooth transitions between different states are available in the visualization [29]. This feature provides the swapping routine with temporary locations between the old and new positions of the parent and child nodes in the tree.

4. Implemented configurations

In Baloian et al [6] we presented several well-known examples for algorithm visualization (AV) using the Algorithm Visualization Using CK application [28]. The examples are classified under themes like sorting, searching, graph algorithms, and string or image processing. They include the well-known algorithms created by Adelson, Velskii and Landis (AVL), Dijkstra, Hoare (QuickSort), Kruskral and Prim (spanning tree) as well as the Bresenham algorithm (see Figure 3).

We have selected algorithms presented in a standard course on Algorithms and Data Structures and have tested the system with a real hardware CK as well as a virtual CK realized on a WACOM graphic tablet and on the screen. In the first stage, the configuration of the CK, the designer must determine the main characteristics of the CK that will be generated. In the second stage, the final user can interact with the CK in order to visualize the algorithms and data structures.

In this paper we focus on the controller design and the use of multimodal interfaces using different perception channels to present the HeapSort algorithm. A slightly different keyboard layout, as shown in Figure 4, can be derived from an implementation by Faltin [18], whose Ph.D. thesis provides a complete analysis of Heapsort and develops an interactive form of learning, programming, simulating and visualizing this algorithm.
Beginning with the actions "insert" and "append" a tree can be built successively (see Figure 4). If the tree has the heap property, the new node is inserted such that the heap property holds for the tree. Otherwise, the new element appears at the end of the tree.

If the heap property is lost, then beginning at the end and going on from right to left, the heap structure is established by comparing (findChild) and exchanging child nodes and their parents (swapUp). Arriving at the root of the tree, the first element of the sorted list is found and exchanged for the last element (moveRoot). The next steps are repeated until the complete list has been sorted. The root element sinks to this correct position (Sink), the heap structure is again reached and the new root is exchanged (moveRoot).

Figure 5 shows a possible second application with the configuration of an overlay intended for the CK. The focus area allows a visually disabled person to feel the positions within the tree. The nodes are labeled with Braille letter signs or rough textures accompanied by speech output. The action selection and data input areas are designed in an analogous way by using tactile symbols. Each action affects the data structure and leads to an acoustic output identifying the new focus, the content of a node and the resulting number of nodes. It should be mentioned that only information on local changes conveys a general understanding of the current tree structure. Finally, this configuration can be complemented by the visualization of the tree discussed above.

5. Related work

We will not review past AV systems referenced in three books on software visualization edited by John Stasko et al [33] and Stephan Diehl et al [11], [14]. Rößling [30] describes methods of generating animation and classifies known systems. There are several repositories of AVs [10], [31] and WWW-pages containing links to collections of AA.
Feiner and McKeown [19] have developed an experimental testbed called COMET (Coordinated Multimedia Explanation Testbed) for the interactive or automated generation of multimedia explanations that combine text and three-dimensional graphics on the fly. In response to a user request for an explanation, COMET decides which information should be expressed in which medium.

The project WIP (Knowledge-Based Presentation of Information, http://www.dfki.de/imedia/wip/) aimed at the development of a presentation system that is able to generate a variety of multimedia documents and to present the same information in different ways, depending on the generation parameters and the individual users in particular communicative situations.

Our work was inspired by M. Eisenberg [17], who offers a number of interface guidelines for mathematical algorithms and suggests ways to provide the user with flexible means for both controlling and understanding the algorithm in question.

In Bridgeman et al [8], a platform-independent e-learning tool, PILOT, was designed; it allows for the generation of random instances of a problem, a user interaction specifying a solution, an evaluation of solutions and a generation of correct solutions to the problem. Our work has points in common with work by Hodes and Katz [21] using XML-documents to configure interfaces and results given by Flippo et al [20] creating multimodal interfaces to control robots for various kinds of tasks using the Java Reflection API. The recently presented MatrixPro system [26] allows the instructor to interact with any data structure already implemented in a library. The framework allows users to create animation sequences and to combine them seamlessly. The main view of the program consists of a menu bar, a toolbar and the area of the visualizations.

Recent work, such as the frameworks Ganimal [12], [13], [16] and LEONARDO [9], uses the event-driven approach to specify the relevant events associated with an algorithm or data structure. Another way to proceed was proposed by Hundhausen [22], [23], who conducted experiments with students who constructed their own AV made out of simple art supplies. In this work, a prototype language and system allows the specification of the AV in terms of spatial logic, postu-
lates a novel interface that supports forward and backward execution and includes a new presentation interface that supports the dynamic markup and modification of an AV. Hundhausen, Douglas, and Stasko [24] classify the experiments that have been done in two categories in order to test the pedagogical benefit of using AVs.

Hundhausen observes that the experiments that allowed students to manipulate the level of involvement had significantly better results than the experiments that allowed them to manipulate the graphical visualization. He concludes that AV software does improve the comprehension level of students and specifically that “what learners do, not what they see, may have the greatest impact on learning” [22]. These observations support our hypothesis that in order to achieve better comprehension the student should be able to participate actively in exploring the algorithm.

Douglas and McLinden [16] use concept keyboard technology to teach early tactile reading. In an educational multimedia application dedicated to blind children, Archambault and Burger [1] report on a touch device as a concept keyboard. Tactile exercise sheets are installed as overlays on the concept keyboard. When the child presses an object placed on an overlay, multimedia events are triggered.

6. Evaluation

Through several phases of evaluation we gained initial indications of the value of the concept and ideas for further development. Interactive CK interfaces for the animation of algorithms offer a great variety of usage scenarios. Attempts to evaluate this new approach have to differentiate between these usage scenarios and clearly define which aspect(s) of the approach to evaluate. For example, we can evaluate the usefulness or usability of the CK configuration process or its implementation in different devices, such as keyboards with different overlays or graphic tablets. Furthermore, the educational benefit of different CK configurations for learners or of other forms of AV ranging to alternative output types, like speech or text, can be examined. On a more detailed level different arrangements of keys on the keyboard or the appropriateness of different earcons may be tested. CKs with tactile overlays combined with acoustic output will also be used in tests in undergraduate courses and schools for the visually impaired.

The new approach was tested threefold. As a preliminary exploratory study at the University of Chile, we designed standardized questionnaires with closed and open questions and exercises for students working with a pre-configured version of the CK and web-based AV software [28]. The results had a great impact on the final implementation of the CK-software. Then the approach was evaluated during two further periods at the University of Duisburg-Essen in January, 2004, and in a more comprehensive test in December, 2004, and January, 2005. The first test group consisted of eighteen students enrolled in different study programs. The system was used to create two kinds of interfaces: a step-by-step keyboard for the Quicksort, and the Kruskal and CKs for the Dijkstra and AVL algorithms - together with two different start-up files for each problem [5]. Then students tried to solve four problems concerning the selected algorithms by hand in a pretest. After intensive study of eight problems concerning Quicksort and Kruskal with the step-by-step keyboard as well as Dijkstra and AVL with the CK, students worked out new problems in a posttest.

Nearly one hundred students with comparable levels of knowledge of A&DS answered a comprehensive questionnaire after having worked with the different implementations of the step-by-step or the CK first designed by the teacher then by the student in a final test. In thirty individual meetings two students tried out the hardware CK and a CK implementation on a Wacom tablet in parallel supervised by a third person. The remaining forty persons repeated the CK software test first undertaken in January, 2004, using more detailed questionnaires and leading to self-configured concept keyboards. The students with less knowledge preferred the step-by-step interface to the CK. This may result from the fact that the step-by-step interface does not require previous understanding of the algorithms. On the other hand, using visualization with the CK led to better results in problem solving than using the step-by-step interface.

The non-parametric Mann-Whitney test was used to examine the answers collected from both groups. We transformed the scores obtained into a ranked list. Then we calculated the test variable U and the ratio \( z = (U - \mu_U) / \sigma_U \). This value is compared to a table of critical values for \( z \) based on the sample size of each group. If \( z \) exceeds the critical value for \( z \) at some significance level (usually 0.05, in our case 0.09), it means that there is evidence for rejecting the null hypothesis in favor of the alternative hypothesis. We conclude that, on the 91% level of significance, the group using the concept keyboard performed better than the group using the step keyboard. The configuration option of the CK was generally appreciated (Median 3.7 on a scale of 1 to 5). Further results are reported in Baloian et al [6].
7. Outlook and further work

We have presented a software architecture for a new controller design that allows users to select a class library on the fly, to instantiate objects and to call their objects. Input and output interfaces are configured through XML files. Next, we will use this concept to implement interactive cryptographic protocols using an interactive Turing machine, private workspaces to generate key words and to encrypt and decrypt messages and shared workspaces to execute the protocol.

Standard control interfaces for algorithm animation were enhanced using appropriate concept keyboards to convey logical structures or data types to human minds. Optional dual or multiple interfaces improve human-machine communication and support sensory-disabled persons. In the future, we want to discuss criteria to select different modalities to transmit the specific output to the learner and to generate cross-modal references on the fly. In a multimodal presentation, cross-modal expressions establish referential relationships of representations in one modality to the ones in another modality as proposed by Wahlster et al [33].

It was an important step to separate control, input and output channels by means of an adequate concept keyboard that can be used to process data structures and geometric models. To achieve a meaningful visualization, we focus on important information after a suitable abstraction process. The concept keyboards can be integrated in existing AA systems. The question remains whether experimenting with algorithms through these new interaction and control elements will lead to a better understanding of how algorithms work and enhance learning benefits from AA technology, especially for the visually impaired.

8. Acknowledgements

This research and common multimedia software development for different learner communities are being carried out by the authors in the course of the recent German-Chilean project ‘Networked Multimedia Systems for Innovative Teaching and Learning Scenarios’ funded by the German ministry BMBF and the Universities of Duisburg and (Santiago de) Chile. The authors would like to thank the referees for their valuable comments.

9. References


WWW-pages containing links to collections of AAs:

http://www.fc-augsburg.de/informatik/projekte/mebib/emiel/emiel_fachgebiert.html

http://www-cg-hci.informatik.uni-oldenburg.de/~da/peters/Kalvin/Doku-UN.htm

http://www.iti.fh-flensburg.de/lang/algorithmen/algo.htm