A Framework for Generating AV Content on-the-fly

Guido Rößling, Tobias Ackermann
Technische Universität Darmstadt
Darmstadt, Germany

roessling@acm.org

Abstract

A main reason for not adopting Algorithm Visualization (AV) for teaching or learning purposes is the time needed to find or generate "appropriate" content (Naps et al. 2003). This paper tries to remedy this by illustrating an easy to use and yet flexible framework for generating content on-the-fly - but tailored to the end user's preferences.

1 Introduction

Most educators and students do not use AV content to improve their teaching or learning. The AV Working Group at ITiCSE 2002 (Naps et al. 2003) examined the underlying reasons for this and gave some guidance on how this could be addressed. In a pre-conference survey, the following major reasons for not adopting AV were given by the survey participants:

- 93% mentioned the "time required to search for good examples",
- 90% mentioned the "time it takes to learn the new tools",
- 90% complained about the "time it takes to develop visualizations",
- 83% mentioned the "lack of effective development tools", and
- 79% stated the time needed to adapt AV content "to teaching approach/course content".

What can be done about this? We can try to make tools easy to use - but this may mean that the time to develop animations will increase. For example, Animal (Rößling and Freisleben 2002) and *JAWAA2* (Akingbade et al. 2003) contain a visual editor for creating AV content by clicking in a GUI front-end. This makes content creation very easy, if also time-consuming. However, ensuring that the portrayed visualization accurately represents the underlying algorithm is difficult. Systems like *Jeliot* (Kannusmäki et al. 2004) require only the appropriate underlying code. However, they are often less prepared for adapting the contents of the visualization to the educator's preferences or course materials.

In this paper, we describe our *generator framework* approach that offers a GUI front-end for user-tailored content and on-the-fly visualization generation.

2 The Generator Framework Approach From the End User's Perspective

We wanted to design a component for easy content generation. The goal of this component was to address the user complaints described above: it simply takes too long to learn a new tool, develop a visualization, or adapt it to the user's preferences. We wanted to address this by reducing most of these complaints:

- Gathering multiple generators in one place, preferably one of the established AV content repositories, makes searching for examples easier;
- The "time to learn" is reduced by using Animal, a system with a relatively simple GUI;
- The "time to develop visualizations" is addressed by using prepared generators, which require just a few mouse clicks;

- As the generators shall create input based on the user's settings, they effectively act as the wanted "effective development tool";
- The "time to adapt content" is reduced by making input values and selected aspects of the visual layout adaptable to the user's wishes.

How does this work? A set of *Generators* are responsible for generating AV content. Typically, each generator will generate a visualization for one concrete algorithm. For example, a *QuickSortGenerator* class may generate a visualization for Quicksort.

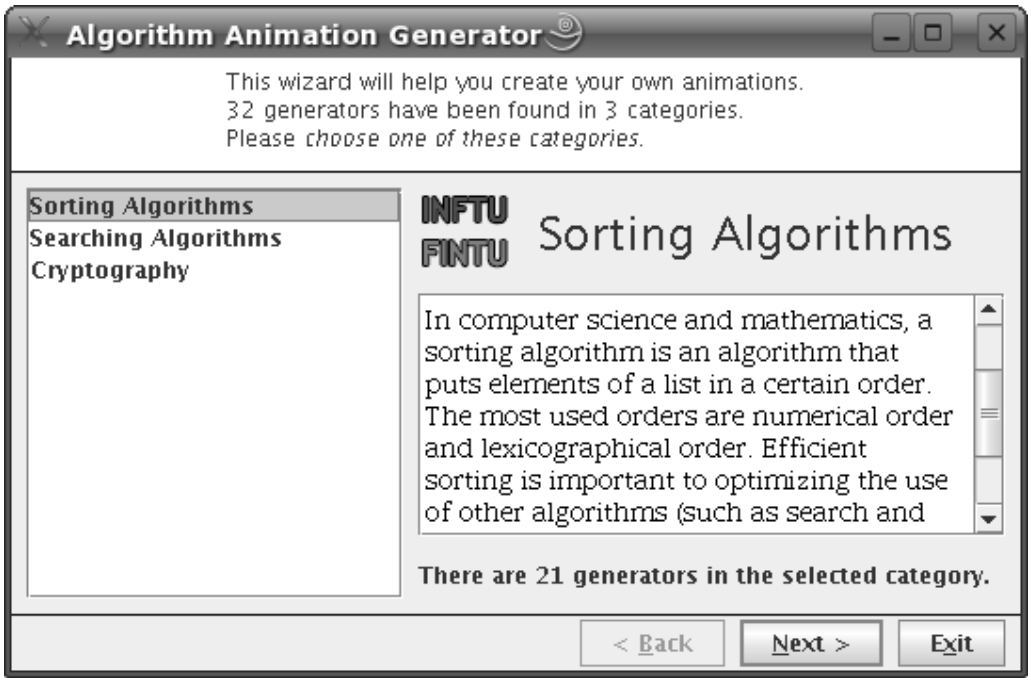

Figure 1: Generator Step 1: Selecting the topic area

The end user first starts up the tool, as shown in Figure 1. The top of the window informs the user about the tool and what steps can be taken next. The user initially sees a list of topic areas (in Figure 1, this is sorting, searching, and cryptography). Once an area is selected, the appropriate icon, title, and description are shown on the right side. At the bottom, the user is informed about the number of content generators available for this category.

After selecting a category, the user can select a concrete generator from a list (Figure 2). The text areas to the right contain a brief description of the generator, and sample output or an example of the underlying code. This is especially important for algorithms such as Quicksort, which exist in a large number of variations. However, only the variant used in the lecture may be helpful for both educators and students. Below these areas, the output format (here "asu" for Animal's AnimalScript format) is shown.

After pressing *Next*, the user is led to the content adaptation step, as shown in Figure 3. Here, the user can edit the input values and selected visual properties. Once the user presses the *Next* button in this step, the framework will ask the user for a filename to which the content shall be stored. After this, the generator is activated and produces the appropriate output, for example, an AnimalScript file for the Animal AV system.

The user can then run the visualization system to show the newly generated visualization, or generate additional content by going back and modifying the values. We have also integrated the generator framework into the Animal AV system, where it is activated by a menu item. Once the user has stored the file, it is automatically parsed by Animal's AnimalScript parser and shown as the new visualization, as displayed in Figure 4.

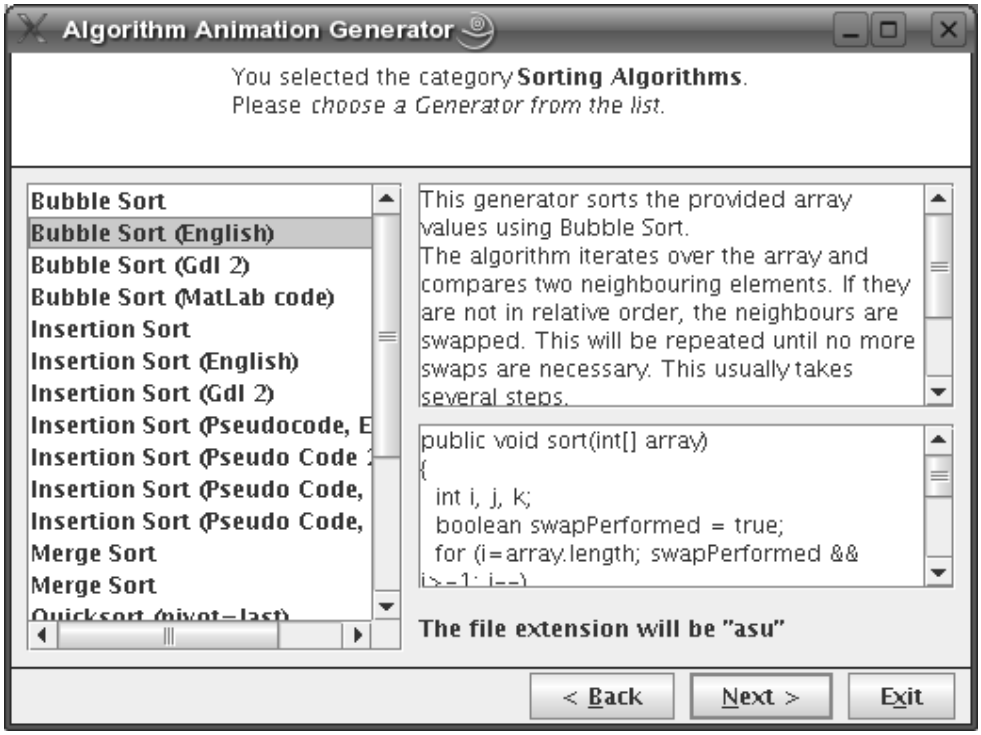

Figure 2: Generator Step 2: Selecting the algorithm

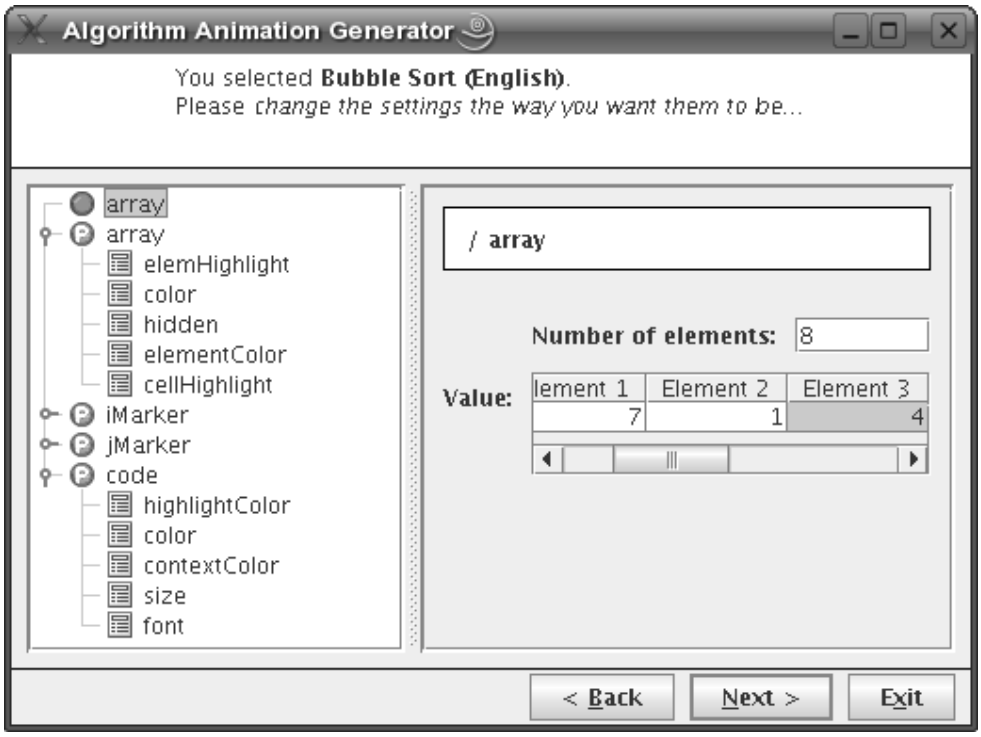

Figure 3: Generator Step 3: Adapting the input values and visual appearance

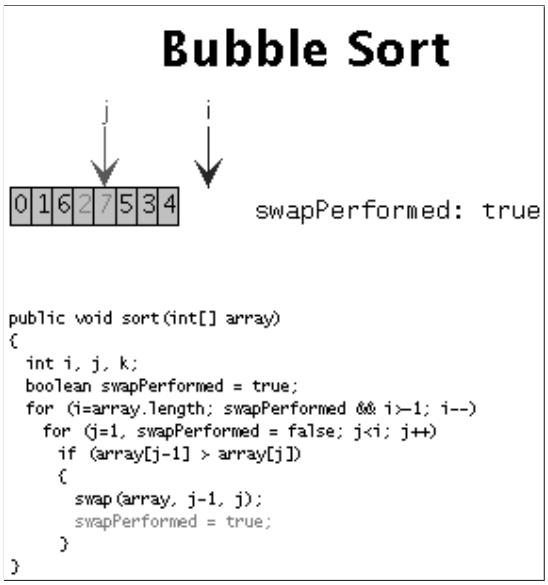

Figure 4: Generator Step 4: The resulting animation

End-users should therefore find it as easy as initially promised to generate their "personal" AV content, including input values and visual properties - assuming, of course, that appropriate content generators are available. In the next section, we will therefore examine the content generator's point of view.

3 The Generator Framework from the AV Content Generator's Perspective

AV content generator implementors have to perform the following simple steps to include a new generator in the generator framework:

1. Implement a generator class for the desired algorithm, placed in the *generatorImplementations* package. The generator has to implement the *generator.Generator* interface. This requires the implementation of the following simple *public* methods:

   GeneratorType getGeneratorType() returns the type of algorithm generated by this algorithm. This information is used to determine the contents of the topic area list in Figure 1. The constructor of class *GeneratorType* expects a combination of the predefined constants for the different sorts of algorithms, e.g., *GeneratorType.GENERATOR_TYPE_SORT*;

   String getName() returns the algorithm's name, as shown in Figure 2;

   String getDescription() returns the algorithm's description (top right of Figure 2);

   String getCodeExample() returns an example of the generator's output (or the underlying algorithm's code); as shown on the lower right side of Figure 2;

   String getFileExtension() returns the file extension for the generator output, e.g., *asu* for AnimalScript, as also shown in Figure 2;

   String generate(AnimationPropertiesContainer props, Hashtable prims) has to generate the content and return it as a String object. The two parameters contain the set of visual properties and the set of user-defined primitive objects, e.g., *int* values or arrays.

   The last method is really the only part that creates some amount of work. The other methods can typically be implemented in less than a minute each.

2. Ensure that the algorithm uses the values specified by the user, not hard-wired settings. This is done by checking that fixed values, e.g. *Color.black*, are replaced by accesses to the *AnimationPropertiesContainer* or *Hashtable* parameters of the *generate* method.

   For example, the array defined by the user in Figure 3 can be accessed as

   ```
   int[] myArray = (int[])prims.get("array");
   ```

   where *prims* is the name of the Hashtable parameter. Similarly, the *highlight* color of the associated code is accessed as

   ```
   Color hlColor = (Color)props.get("code", "highlightColor");
   ```

   where *props* is the AnimationPropertiesContainer parameter passed to the *generate* method. Note that "array" and "code" match the name of the array primitive and the code property set in Figure 3, respectively. "highlightColor" is the name of one of the properties for the "code" property set.

3. Specify the properties and primitives used by the algorithm. A simple GUI front-end can be used for this, as shown in Figure 5. This front-end lets the author select the set of primitives and properties used by the generator.

   Each primitive and property has a default value which can be adjusted according to the author's preferences. Additionally, the author can decide individually for each property whether this should be user-editable. Thus, the end user will be able to adapt anything between "nothing" and "everything" regarding the visual layout to his or her tastes.

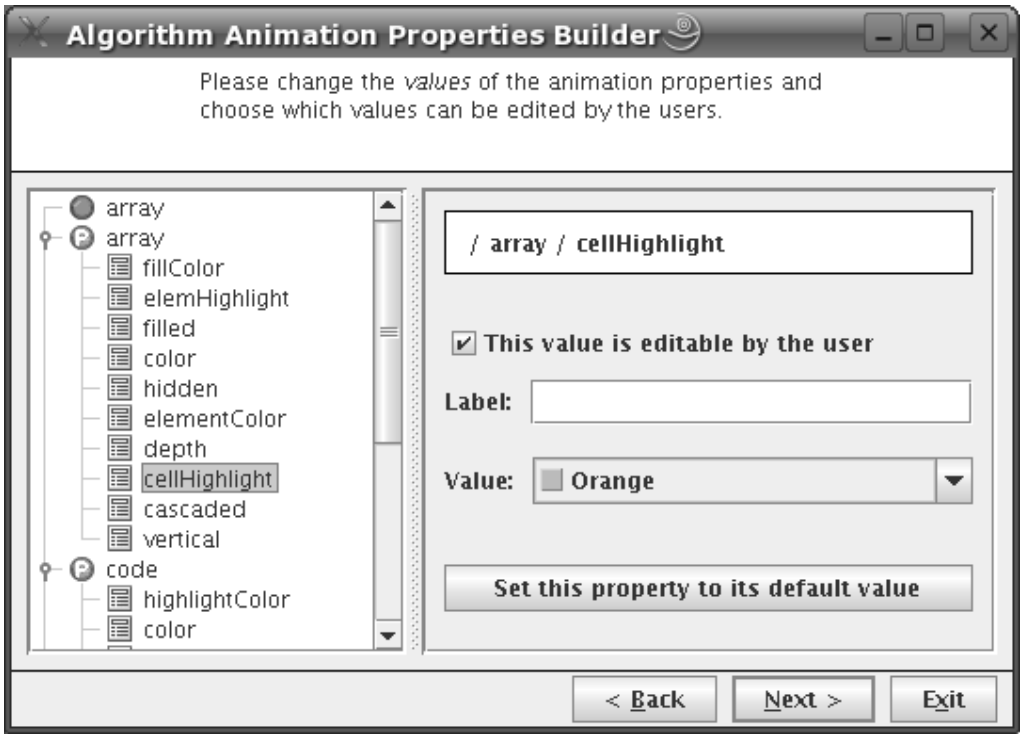

Figure 5: The Animation Properties Builder front-end for content generation authors

Figure 5 shows the properties generation window. The first frame (not shown in the Figure) allows the content generation author to load an existing specification or create a new one. Each specification can contain a set of folders, primitive objects, and properties (indicated by the circled letter P). The editing front-end for primitives is identical to the one shown in Figure 3.

As shown in Figure 5, the author can specify whether a given entry – here, "cellHighlight" – shall be editable by the end-user. When comparing Figures 3 and 5, it becomes

obvious that the author has deactivated user editing for a number of properties, such as *fillColor, filled, depth, cascaded,* and *vertical* for the "array" element.

Once the editing process is completed, the settings are stored as an XML file.

The implementation of the generator, together with the XML specification, drive the generation process shown in the first three Figures of this paper. From our experience, about 80%-95% of the time needed to implement an algorithm generator are tied to the *generate* method, and therefore to the actual visualization. This leaves only an overhead of 5-20% for embedding the generator into the framework - and in most cases, it will probably be 20% for only the first attempt, and closer to 5% for all others.

At the moment, there are more than 30 generators for the topic areas sorting, searching, and cryptography, as indicated in Figure 1. We hope to increase this number even further before PVW 2006, and plan to add other areas, such as tree algorithms and string searching.

The generator framework is fully internationalized, although it currently only supports English and German. Volunteers who want to help in translating the language resources (less than 100 lines of text for the GUI) are welcome!

4 Summary and Further Work

In this paper, we have presented a simple yet expressive framework that makes it significantly easier for an AV end-user to specify the input for a visualization - and adapt its visual properties to the user's preferences or content design. Using the component is easy and straightforward for the end-user, and also easy and quick for the AV content generator. The component can be integrated easily into existing systems, as shown in the example integration into the Animal system.

We hope that the generator framework will be intensively discussed during the Workshop. Although the framework has only been used in conjunction with Animal so far, there is no reason why it should not be able to produce output for other AV systems, such as JAWAA2 (Akingbade et al. 2003).

Additionally, we hope that some of the other AV system authors will want to consider adopting (or adapting) this framework to suit their personal tastes - and their own system.

References

1 Ayonike Akingbade, Thomas Finley, Diana Jackson, Pretesh Patel, and Susan H. Rodger. JAWAA: Easy Web-Based Animation from CS 0 to Advanced CS Courses. In *Proceedings of the* 34^{th} *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2003), Reno, Nevada*, pages 162–166. ACM Press, New York, 2003.

2 Osku Kannusmäki, Andrés Moreno, Niko Myller, and Erkki Sutinen. What a Novice Wants: Students Using Program Visualization in Distance Programming Courses. In *Proceedings of the Third Program Visualization Workshop, Research Report CS-RR-407*, pages 126–133. Department of Computer Science, University of Warwick, UK, 2004.

3 Thomas L. Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, and J. Ángel Velázquez-Iturbide. Exploring the Role of Visualization and Engagement in Computer Science Education. *ACM SIGCSE Bulletin*, 35(2):131–152, 2003.

4 Guido Rößling and Bernd Freisleben. Animal: A System for Supporting Multiple Roles in Algorithm Animation. *Journal of Visual Languages and Computing*, 13(3):341–354, 2002.