Alcom-FT Technical Report Series
ALCOMFT-TR-03-49

A System for Building Animated Presentations over the Web *

Vincenzo Bonifaci † *Benedetto A. Colombo* ‡ *Camil Demetrescu* § *Irene Finocchi* ¶ *Luigi Laura* ‖

Abstract

We describe Leonardo Web, a collection of tools for building animated presentations that can be useful for teaching, disseminating, and e-learning. Presentations can be created via the combined use of a visual editor and a Java library. The library allows it to generate animations in a batch fashion directly from Java code according to an imperative specification style. Batch-generated animations can then be refined and customized using the editor. Presentations can be finally viewed with a simple Java player, which ships both as a stand-alone application for off-line deployment and as a Java applet embedded in a Web page. The player supports step-by-step and continuous execution, reversibility, speed selection, and smooth animation.

*This work has been partially supported by the IST Programme of the EU under contract n. IST-1999-14.186 (ALCOM-FT) and by the Italian Ministry of University and Research (Project "ALINWEB: Algorithms for Internet and the Web").

†Dipartimento di Informatica e Sistemistica, Università degli Studi di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy. Email: `Vincenzo.Bonifaci@dis.uniroma1.it`.

‡Dipartimento di Informatica e Sistemistica, Università degli Studi di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy. Email: `ba.colombo@tiscali.it`.

§Dipartimento di Informatica e Sistemistica, Università degli Studi di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy. Email: `demetres@dis.uniroma1.it`. URL: `http://www.dis.uniroma1.it/~demetres`.

¶Dipartimento di Informatica, Sistemi e Produzione, Università degli Studi di Roma "Tor Vergata", Via di Tor Vergata 110, 00133 Roma, Italy. Email: `finocchi@disp.uniroma2.it`. URL: `http://www.disp.uniroma2.it/users/finocchi/`.

‖Dipartimento di Informatica e Sistemistica, Università degli Studi di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy. Email: `laura@dis.uniroma1.it`. URL: `http://www.dis.uniroma1.it/~laura`.

1 Introduction

Interactive animations are a valuable tool for teaching and learning: they can be used by algorithm researchers who want to share and disseminate their ideas, by lecturers to liven up lectures, to demonstrate the behavior of complex systems, or to portray the dynamic aspects of some topic of interest, and by students for individual experiments, so as to deepen the knowledge acquired in the lectures. Realizing illustrative computer-based learning material is usually very expensive [4, 8], and the task is even more cumbersome if one wants animations to be portable across different platforms. Instructors who wish to prepare animated presentations are typically required to use some commercial general-purpose tool, or even to write adhoc computer programs: this might be difficult and time-consuming. For instance, using tools such as Microsoft PowerPoint, which provides a flexible support for creating animated slides, may be hard if complex animations are to be produced. On the other hand, specialized tools, such as Macromedia Flash, can be used to create highly customized animations, but may have a steep learning curve. Creating animated GIFs of MPEG movies might be even harder and limits the user interaction possibilities. In this context, exploiting Web-based technologies for education seems to be a quite natural solution [3, 10, 14, 16]. However, as detailed in Section 2, the quest for simple, light, and easy-to-use tools for building general-purpose animated presentations over the Web still demands for further efforts.

In this paper we describe Leonardo Web, a collection of Web-based tools for creating animations that can be useful for teaching, disseminating, and e-learning. Animated presentations can be created with a specialized visual editor and viewed with a simple Java player, which is available both as a stand-alone application for off-line deployment and as a Java applet. The player supports step-by-step and continuous execution of animations, which can be run both forward and backward at different speeds. To support visualization of algorithmic concepts, Leonardo Web also provides a library that can be used to generate animations directly from Java algorithm implementations according to an imperative style, i.e., by inserting calls to graphical routines in the points of the code where the events of interest take place. Presentations created with Leonardo Web, which include text, 2D graphics, and bitmapped images with smooth animation effects, ship as plain text files written in a simple scripting language. Animation scripts are small and compact, and can specify highly complex graphical scenes. Batch-generated animations can be easily refined and customized using the editor. Leonardo Web is written in Java and is made of three main components:

- *The Builder*: a visual editor, which can be used to build and edit presentations;
- *The Player*: a viewer for Leonardo Web presentations, which can be used both as a Java stand-alone application and over the Web as a Java applet;
- *The Library*: a Java library that supports creation of batch-generated animation scripts using Java programs as drivers. Animations created in this way can be further refined using the Builder.

Up to date information about Leonardo Web tools can be found at the Web site `http://www.dis.uniroma1.it/~leoweb`.

The rest of this paper is organized as follows. In Section 2 we discuss the most common approaches to creating animations over the Web and present recent related work. In Section 3 we describe the main components of Leonardo Web, discussing their main features and the overall design of the system. In Section 5 we address different scenarios that show how Leonardo Web can be used to create presentations for educational and dissemination purposes. Section 6 gives concluding remarks and discusses directions for future work.

2 Related work

Despite the impressive rise of Web authoring applications in the last years, creating animated presentations as a support for e-learning and teaching over the Web still remains a challenging task. The most popular trend is to write ad-hoc Java applets that display the desired animations (see, e.g., [1, 13, 17, 22]). With this technique, highly customized and interactive presentations can be obtained, but preparing them may be long and boring. Creating presentations of computer science concepts has motivated researchers to think about ways to automate (or at least to simplify) the process of visualizing, e.g., how programs and algorithms work. In particular, a few Web-based systems can be found in the literature.

JEliot [11] automatically produces visualizations of Java programs by parsing the Java code and allowing the user to choose a subset of variables to visualize on the stage according to built-in graphical interpretations. It relieves the user from writing any visualization code and is very easy to use, but lacks in customization possibilities and abstraction. Thus, it is mostly useful to illustrate basic programming concepts. JDSL [2] is a Java library of data structures that features a visualizer for animating operations on abstract data types such as AVL trees, heaps, and red-black trees [6]. It is well suited for educational purposes, as students are allowed to write and test their own classes provided they implement specific JDSL Java interfaces. The visualization of supported data types, however, is embedded into the library and cannot be changed. VEGA [12] is a C++ client/server visualization environment especially targeted to portraying geometric algorithms: while the algorithm is executed on the server, the clients runs on any Java Virtual Machine and a small bandwidth communication interface guarantees good performance even on slow networks. The end-user can visualize algorithms on-line or show saved runs off-line, and can customize the visualization by specifying a suitable set of view attributes. WAVE [7] is an algorithm visualization tool based on a publication-driven approach: algorithms run on a developer's remote server and their data structures are published on blackboards held by the clients. Animations are specified by attaching visualization handlers to the data structures published on the client's blackboard: modifications to these structures, due to the remote algorithm execution, trigger the running of the corresponding handlers on the client's side. Other Web-based algorithm animation tools are mentioned in [9, 21].

In order to realize an animated presentation, all those systems hinge upon the existence of an underlying running program and are tied to a specific programming language. The Samba package [19], and its Web-based follow up JSamba, represent a first effort towards a language-independent solution to algorithm animation. Samba provides an interpreted front-end to the Tango system [20] and uses a scripting-based approach: it reads an ASCII file, one command per line, in order to acquire directions for creating an animation. Thus, an algorithm can be easily visualized by placing print statements in the underlying program, which can be implemented in any language. Unfortunately, this often requires to take into account very low-level details about the visualization, e.g., explicitly specifying the position of objects in the graphical scene: wrapping the print statements into calls to methods of a more abstract library would instead hide some of these tedious details.

In many cases it would be quite useful to be able to animate high-level concepts, independently of a specific algorithm, such as, e.g., rotations in balanced binary search trees. In this scenario, starting from an underlying program may be difficult, and a visual editor to animate these proofs of concepts would be much useful. A recent release of the JAWAA system [15] exploits the use of an editor to generate animation traces in a scripting language similar to that of Samba. However, it does not support integration of the editor capabilities with the traditional program-driven animation approach. Indeed, creating a presentation may be long and boring using a visual editor only, which in turn may be useful to refine a visualization skeleton by adding comments, explanations, and by orchestrating the overall graphical layout. The ANIMAL system [18] was designed to combine visual editing with batch generation. Unfortunately, the system, which provides powerful features for creating lecture presentations, seems to be unavailable for Web-based deployment.

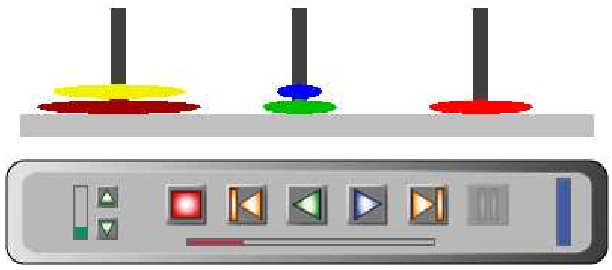

Figure 1. The Leonardo Web Player showing the Hanoi Towers Challenge.

3 An Overview of Leonardo Web

Leonardo Web is a collection of tools for building and viewing animated presentations. The system is written in Java and uses both stand-alone and applet technologies, allowing users to create and view presentations off-line, and then easily post them over the Web for remote access. In this section we describe the main features of the Leonardo Web tools, addressing the key aspects of our approach.

3.1 The Builder

Presentations in Leonardo Web can be easily created using the Builder, a visual editor for building animations (we refer to Figure 2 for a snapshot). The tool allows the user to write and maintain a sequence of key frames, which are the backbone of the presentation. Each frame can contain text and 2D graphical objects drawn from a vocabulary of elementary geometric shapes, including circles, ellipses, lines, and rectangles. User-defined bitmapped images can also be added to the graphical scene. The user can interact with the Builder's GUI in order to add, resize, move, and delete graphical objects. The sequence of key frames in the presentation is shown in a window as a list of numbered *thumbnails*, which allow the user to control the big picture of the presentation and select individual frames for editing. Each graphical object in a scene is assigned an identification number and a set of attributes, including size, position, and color, which can be individually modified using the object inspector window. To support smooth animation, objects by the same identification number in consecutive scenes are compared, and attribute changes are interpolated to form a sequence of intermediate frames. The number of intermediate frames is an attribute of the object itself, and can be fully controlled by the user.

The Builder can save a presentation as a plain text file written in a simple *scripting language*, which specifies the incremental changes that lead from a key frame to the successive one. The benefits of using scripting languages for generating program visualizations have been pointed out by several authors [15, 18, 19]. In particular, our language was inspired by that of Samba [19] and JAWAA [15]. The incremental nature of our scripting language makes presentation scripts small and compact, and thus amenable to quick download even on slow network connections.

Presentations created with the Builder can be visualized with the Player, as we will see in Section 3.2, and can be later reopened with the Builder itself, which reconstructs the sequence of key frames, for additional editing. Interestingly, the Builder can also open and modify presentations created in some other way (e.g., directly writing a script or using the Library): this allows the users to create and refine presentations via the combined use of different tools, using the most appropriate one in different stages of the animation specification process.

The Builder is currently being developed and tested on different platforms: up to date information can be found at the Web site `http://www.dis.uniroma1.it/~leoweb`.

3.2 The Player

The Leonardo Web Player has been designed as a light and easy-to-use presentation viewer. It is able to interpret text files created by the Builder, and can be used both as a stand-alone Java application and as an applet inside a Web page. The graphic user interface of the Player, shown in Figure 1, is clean and simple, resembling to a standard VCR control tool. The user can start, stop, rewind, and play the presentation both forward and backward. Playing is supported either in a step-by-step fashion, or continuously. Animated transitions of graphical objects, including movements and color changes, are smoothly rendered by the system by generating sequences of interpolating interframes.

To support effective on-line deployment even on slow network connections, the Player is fully multithreaded, allowing it to start playing a presentation even if it has not been completely downloaded from the remote peer. Furthermore, the applet version of the Player is just about 90KB, including GUI graphics. This imposes a light burden on the applet startup phase, which is typically very time-critical. The interested reader can see the Leonardo Web Player in action at the Web site: `http://www.dis.uniroma1.it/~leoweb`.

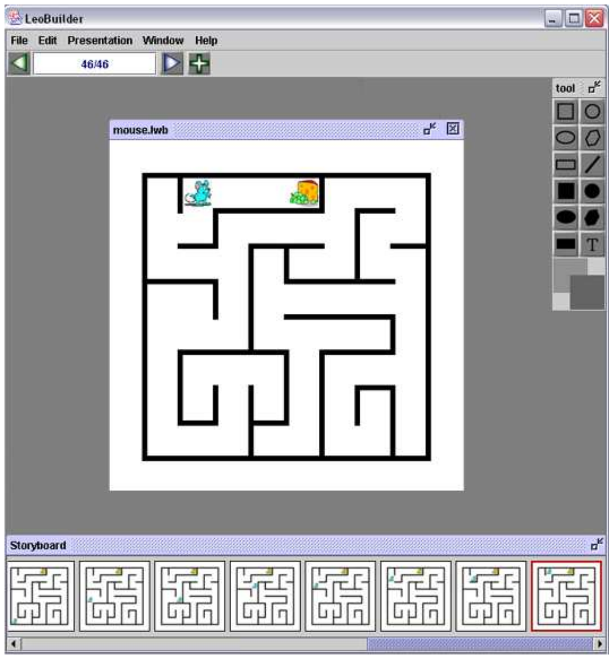

Figure 2. Using the Builder to prepare an animated presentation.

3.3 The Library

While the Builder appears to be flexible enough to support common user's needs, sometimes presentations include complex animations that portray some technical aspect of a topic of interest, which might be difficult to specify visually. In this scenario, it might be easier to write a program whose execution produces the desired animation script, rather than having to specify it directly in the Builder. Still, a script generated in this way can be later refined and completed using the Builder. To support this scenario, which will be addressed in more depth in Section 5, Leonardo Web

```
public class Bouncing {
  public static void main(String[] args){
    int x=0, y=0, dx=1, dy=1;
    JLeoScript s = new JLeoScript("bouncing.lwb");
    s.newCircle("ball", 0, 0, 10, 200, 0, 0, true);
    s.newRectangle("box", 0, 0, 400, 200, 160, 160, 160, false);
    for (i=0; i<200; i++) {
      if (x==0 && dx<0 || x==14 && dx>0) dx=-dx;
      if (y==0 && dy<0 || y==9  && dy>0) dy=-dy;
      x+=dx; y+=dy;
      s.moveAbsolute("ball", x*20, y*20, 4);
    }
    s.close();
  }
}
```

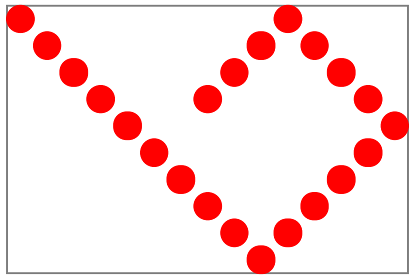

Figure 3. Program that generates a "bouncing ball" animation script.

provides a Java library (`JLeoScript`) that provides primitives for creating presentation scripts.

Available primitives supported by the class `JLeoScript` include adding graphical objects to the scene, deleting existing graphical objects, moving and resizing objects, and changing color. As an example, in Figure 3 we show a Java program that uses the `JLeoScript` library to generate a simple animated presentation. The produced script simulates a ball bouncing in a box as shown in the same figure. To achieve this goal, the program first creates a `JLeoScript` object, specifying the name of the file to be created, and then adds the ball (`newCircle`) and the box (`newRectangle`) to the scene. Those graphical objects are defined by specifying a unique name (e.g., "ball", "box"), the coordinates of the left-top corner, the object width and height, the color in RGB format, and a flag telling whether the object has to be color-filled. The program then enters a simulation loop that lets the ball bounce inside the box (`moveAbsolute`). Each elementary movement of the ball is smoothly interpolated using 4 interframes.

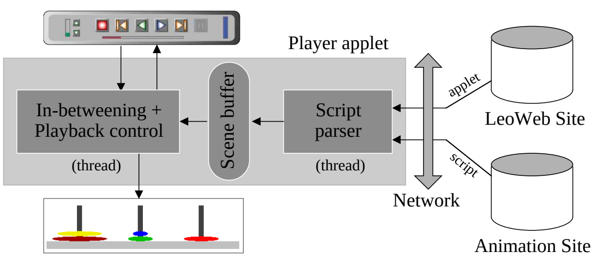

Figure 4. Internal Organization of the Player.

4 Architecture of the System

In this section we briefly describe the internal architecture of the Player. Details about the Builder will be given in the full version of this paper. The structure of the Player is shown in Figure 4. Two threads cooperate in playing an animation script: a language parser and an interactive animation module. As the applet is loaded from the Leonardo Web site, the language parser establishes a connection to the remote site that contains the desired script, which is loaded and parsed. In the off-line application version of the Player, the language parser retrieves the script from a local disk, rather than from a Web site. The language parser produces an indexed sequence of scenes, each of which consists of a list of data records describing operations on graphical objects. The interactive animation module is activated by the user through the buttons in the control palette. This module is able to interpret the operations in the scene buffer according to the playback direction, incrementally creating interpolating interframes for smooth animation rendering.

5 Leonardo Web in Action

In this section we show how Leonardo Web can be used to prepare a presentation and how animations can be made available over the Web.

5.1 Preparing a Presentation

We consider different usage scenarios, detailing how the different parts of the system can be effectively exploited to achieve the desired result.

```
public class BubbleSort {
    public static void main(String[] args){
        int[] v = { 3, 7, 2, 5, 9, 1, 4, 8, 6 };
        bubblesort(v);
    }
    private static void bubblesort(int[] v){
        boolean finished;

        do {
            finished = true;
            for (int i=0; i<v.length-1; i++)
                if (v[i]>v[i+1]) {

                    int temp = v[i];
                    v[i]     = v[i+1];
                    v[i+1]   = temp;
                    finished = false;
                }
        } while(!finished);

    }
}
```

```
JLeoScript s = new JLeoScript("bubble.lwb");
s.begin();
for(int i=0; i<v.length; i++)
   s.newRectangle(v[i],20+i*20,20,18,20*v[i],
                  180,180,180,true);
s.end();
```

```
s.begin();
s.moveRelative(v[i],20,0,7);
s.moveRelative(v[i+1],-20,0,7);
s.end();
```

```
s.close();
```

Figure 5. Java implementation of the Bubblesort algorithm.

Using the Builder. Visual editors may be especially useful to prepare simple animations that illustrate high-level concepts (e.g., rotations in balanced binary search trees), as well as to refine more sophisticated presentations obtained from program execution traces. Indeed, as noted by Brown and Hershberger, "even though it may be easy to animate a program, it's not so easy to produce an effective and informative visualization" [4]. This requires adding comments, labels, and a lot of text explanations that are typically quite boring to be realized. An appealing graphical layout should be orchestrated, other essential graphical features, such as text color and size, should be customized, and their meaning explained. In this context, we believe that an integrated use of the Library and the Builder can be very beneficial.

In the following, we briefly describe a usage example of the Builder related to the first scenario. Let us assume that we want to illustrate the idea behind backtracking by means of a simple toy example: a mouse in a labyrinth is seeking for its cheese. The mouse moves along a path in the labyrinth until either a dead-end street or the cheese is found. In case of a dead-end street, the mouse backtracks to the nearest branching and takes a different path. An animation like this can be easily prepared using the Builder. The labyrinth is made up of vertical and horizontal lines, which can be added by means of the object inspector window. Cheese and mouse can be represented using two bitmapped images. When a scene is ready, we can commit it and then obtain the successive one by incremental modification (i.e., we just need to reposition the mouse). The graphical storyboard allows us to select any scene, modify it, add a new scene or delete an existing one. For instance, if we decide to change the structure of the labyrinth, it suffices to modify the starting scene and propagate the change to all the successive ones. A snapshot of the Builder in use is illustrated in Figure 2 and the animation is available from the Leonardo Web site [5].

Similarly, the Builder can be used in order to refine an existing animation. In this case, when an existing presentation is opened, the key frames are parsed and appear in the graphical storyboard. Any scene can then be modified, possibly propagating changes to a subset of the subsequent scenes, and the new animation can then be saved and played as usual.

Using the Library. Creating a presentation might be long and boring even using a visual editor. This is especially true if the content is complex and technical. For instance, creating a presentation to explain how an algorithm works on some input data for a Computer Science class might be hard and even error-prone due to the difficulty to capture all the aspects of the algorithm execution. A classical approach in algorithm animation [21] consists of using an implementation of the algorithm to be visualized as a driver, and let it emit a sequence of events which are turned into graph-

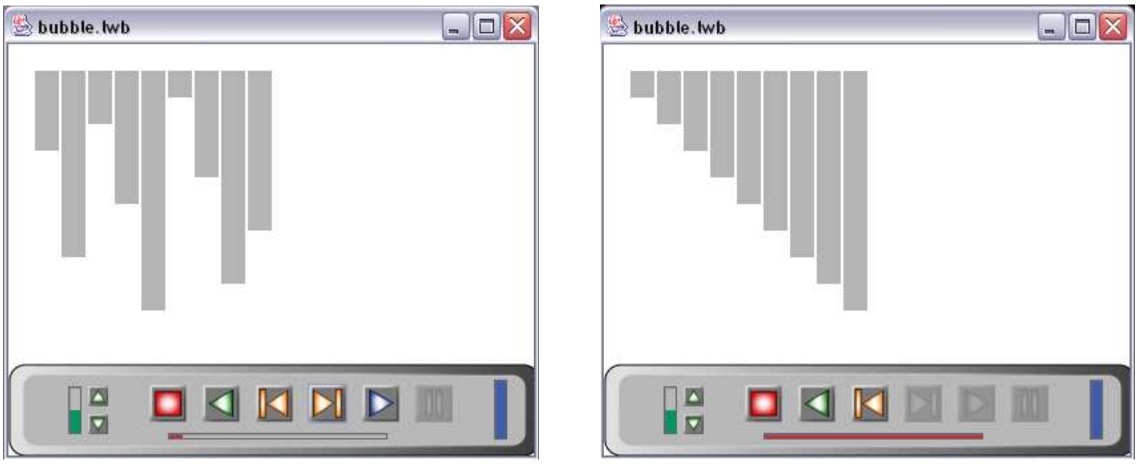

Figure 6. Screenshots of the bubblesort visualization.

ical commands as the implementation runs. This is typically achieved by annotating the algorithm implementation with suitable calls to library routines.

To explain how this approach can be supported using Leonardo Web, let us consider a concrete example. Suppose that an instructor has to prepare a presentation showing the Bubblesort algorithm in action. The starting point will be a Java implementation of the algorithm like the one shown in Figure 5.

A classical visual metaphore for portraying sorting problems displays the array as a sequence of sticks with height proportional to the array values [21]. Swaps of items are illustrated by exchanging the positions of the corresponding sticks, and the final sorted array is displayed as a growing sequence of sticks. The instructor can annotate the code using the Leonardo Web Library as shown in the grey boxes in Figure 5. Running the annotated program produces the desired animation script, which is shown in Figure 6.

With this approach, the same implementation can be used to prepare different animated presentations of the same algorithm on a variety of data sets in order to show, for instance, the behavior of the algorithm on worst-case instances. Of course, animations produced in this way may be further refined, e.g., by adding labels, captions, and other information to focus specific aspects of the algorithm execution.

5.2 Delivering a Presentation over the Web

Using the applet version of the Player, it is easy to set up Web pages that include animations created with Leonardo Web. To explain how this can be done, we consider again a concrete example. Suppose that the instructor wants to create a Web page that shows the bubblesort animation described in Section 5.1, so that students can view it. To this aim, the only thing she has to do is to drop the animation script file `bubble.lwb` in her website together with a Web page containing an html tag like the one shown in Figure 7.

When students access the page, the Player applet is loaded, and the bubblesort animation script is automatically fetched by the applet, allowing students to view it. Notice that the instructor does not even have to install the Player in her Web site, since the Web browsers used by the students will fetch it directly from our site `http://www.dis.uniroma1.it/~leoweb`.

6 Conclusions and Work in Progress

In this paper we have presented Leonardo Web, a collection of tools for creating and viewing animated presentations. Animations generated with Leonardo Web include both text and 2D graphics with smooth animation effects, and ship as compact text files written in a simple scripting language. The system is written in Java and includes a visual editor for editing presentations (the Builder),

```
<applet codebase =
 http://www.dis.uniroma1.it/~leoweb/jlwCode/
 code     = jLeoWeb.JLeoWeb
 archive  = "jLeoWeb.jar"
 width    = 400 height = 470>
<param name  = "file" value = "bubble.lwb">
<param name  = "gui"  value =
 "http://www.dis.uniroma1.it/~leoweb/jlwGui">
</applet>
```

Figure 7. HTML tag for including the `bubble.lwb` presentation in a Web page.

a presentation viewer (the Player), and a Java library for creating batch-generated animation scripts (the Library). Presentations can be easily posted over the Web using the applet version of the Player. Leonardo Web is freely available over the internet at `http://www.dis.uniroma1.it/~leoweb`. Differently from previous Web-based systems such as JAWAA [15], Leonardo Web explores integration of visual editing and batch generation. As another relevant feature, the Player supports going back to previous animation frames. Without this feature, which is rarely implemented, the user has to restart the animation from the beginning if a transition of interest is passed over, strongly limiting the effectiveness of the presentation.

We are currently extending the system in several directions. In particular, we remark that the Builder is still in a prototype stage, and new features are currently being added. To allow interested people to modify and extend the system, we plan to distribute the source code of Leonardo Web under the terms of the GNU General Public License (GPL). Furthermore, we believe that writing converters that generate PostScript versions of the animations would be a valuable contribution for authors who wish to include storyboards in their papers. Another useful addition would be exporting Leonardo Web presentations in a format readable by standard tools such as Microsoft PowerPoint.

References

[1] Algorithma. Department of Computer Science, California State University, 2000. URL: `http://web.csusb.edu/public/class/cs455_1/winter2000/index.html`.

[2] R.S. Baker, M. Boilen, M.T. Goodrich, R. Tamassia, and B. Stibel. Testers and Visualizers for Teaching Data Structures. *SIGCSE Bulletin (ACM Special Interest Group on Computer Science Education)*, 31, 1999.

[3] C.M. Boroni, F.W. Goosey, M.T. Grinder, and R.J. Ross. A Paradigm Shift! The Internet, The Web, Browsers, Java, and the Future of Computer Science Education. *SIGCSE Bulletin: Proc. 29th SIGCSE Technical Symposium on Computer Science Education*, 30(1):145–149, 1998.

[4] M.H. Brown and J. Hershberger. Color and Sound in Algorithm Animation. *IEEE Computer*, 25:52–63, 1992.

[5] B.A. Colombo, C. Demetrescu, I. Finocchi, and L. Laura. Leonardo Web site, 2003. `http://www.dis.uniroma1.it/~leoweb`.

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill, 2001.

[7] C. Demetrescu, I. Finocchi, and G. Liotta. Visualizing Algorithms over the Web with the Publication-driven Approach. In *Proc. of the 4-th Workshop on Algorithm Engineering (WAE'00)*, LNCS 1982, pages 147–158, 2000.

[8] A. Diaz de Ilarraza Sanchez and I. Fernandez de Castro, editors. *Proceedings of the 3rd Int. Conference on Computer-Aided Learning and Instruction in Science and Engineering*, Spain, July 1996.

[9] S. Diehl, editor. *Software Visualization*. LNCS 2269. Springer Verlag, 2001.

[10] J. Domingue and P. Mulholland. An Effective Web Based Software Visualization Learning Environment. *Journal of Visual Languages and Computing*, 9(5):485–508, 1998.

[11] J. Haajanen, M. Pesonius, E. Sutinen, J. Tarhio, T. Teräsvirta, and P. Vanninen. Animation of User Algorithms on the Web. In *Proceedings of the 13th IEEE International Symposium on Visual Languages (VL'97)*, pages 360–367, 1997.

[12] C.A. Hipke and S. Schuierer. VEGA: A User Centered Approach to the Distributed Visualization of Geometric Algorithms. In *Proceedings of the 7-th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media (WSCG'99)*, pages 110–117, 1999.

[13] L. Kucera. Homepage. URL: `http://www.ms.mff.cuni.cz/acad/kam/kucera`.

[14] T. Naps. Algorithm Visualization Served Off the World Wide Web: Why and How. *ACM SIGCSE Bulletin*, 28:66–71, 1996.

[15] W.C. Pierson and S.H. Rodger. Web-based Animations of Data Structures Using JAWAA. In *Proc. 29th SIGCSE Technical Symposium on Computer Science Education*, pages 267–271, 1998.

[16] R.J. Ross and M.T. Grinder. Hypertextbooks: Animated, Active Learning, Comprehensive Teaching and Learning Resources for the Web. In S. Diehl, editor, *Software Visualization*, LNCS 2269, pages 269–284. Springer Verlag, 2001.

[17] G. Rößling. Collection of animations. URL: `http://www.animal.ahrgr.de/`.

[18] G. Rößling, M. Schüler, and B. Freisleben. The Animal Algorithm Animation Tool. In *Proceedings of the 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education (ITiCSE 2000)*, pages 37–40, 2000.

[19] J.T. Stasko. Algorithm Animation Research at GVU. `http://www.cc.gatech.edu/gvu/softviz/algoanim/`.

[20] J.T. Stasko. TANGO: A Framework and System for Algorithm Animation. *IEEE Computer*, 23:27–39, 1990.

[21] J.T. Stasko, J. Domingue, M.H. Brown, and B.A. Price. *Software Visualization: Programming as a Multimedia Experience*. MIT Press, Cambridge, MA, 1997.

[22] M. Syrjakow, J. Berdux, and H. Szczerbicka. Interactive Web-based Animations for Teaching and Learning. In *Proceedings of the 32nd Winter Simulation Conference*, pages 1651–1659. Society for Computer Simulation International, 2000.