A Cross-Domain Visual Learning Engine for Interactive Generation of Instructional Materials

K.R. Subramanian
Department of Computer Science
The University of North Carolina at Charlotte
Charlotte, NC 28223, USA
krs@uncc.edu

T. Cassen
Department of Computer Science
The University of North Carolina at Charlotte
Charlotte, NC 28223, USA
bspt@pipeline.com

ABSTRACT

We present the design and development of a *Visual Learning Engine*, a tool that can form the basis for interactive development of visually rich teaching and learning modules across multiple disciplines. The engine has three key features that makes it powerful and cross-disciplinary, (1) it is based on a *finite state machine* model, that supports concepts presented in any defined sequence, (2) instructional modules are designed and generated *interactively* using graphical interface widgets, facilitating non-programmers to use the system, and (3) ability to simultaneously present *concepts and their visual representation* that allows for a more intuitive and exploratory learning experience. We demonstrate a prototype of the learning engine by testing it on examples from Computer Science(sorting algorithms, recursion) and Electrical Engineering (signal manipulations).

Categories and Subject Descriptors

E.1 [**Data Structures**]: Arrays; I3.7 [**Computer Graphics**]: Animation; K3.1 [**Computer Uses in Education**]: Computer Assisted Instruction.

General Terms

Algorithms

Keywords

finite state machine, algorithm, signal, cross-disciplinary

1. INTRODUCTION

Most disciplines in science and engineering involve *core foundation-building technical* courses, especially at the freshman/sophomore levels, that can pose the biggest stumbling blocks to undergraduate student learning. These courses involve learning *fundamental concepts* in challenging environments, such as large class sizes, graduate student or temporary instructors distributed across multiple course sections,

etc. This can result in long-term negative impacts, making it unattractive to enter certain disciplines, and have an impact on retention of discipline majors.

Over the last decade, desktop computer technology has undergone a major revolution: today's workstations are extremely powerful, are readily accessible to students and support multimedia technologies. These features make it possible to consider new ways of providing learning tools that are *highly scalable*. In particular, the design and use of interactive learning tools using a combination of *graphics, animation and visualization* is very much feasible in today's learning environments.

In this article, we present the design and development of a *Visual Learning Engine*, that can form the infrastructure for rapid creation of *new learning and teaching modules across multiple disciplines*. The key ideas that make this general enough to be utilized across multiple disciplines are as follows:

1. The basic kernel of our design is based on a *Finite State Machine(FSM)*, a model that universally supports concepts or modules that can be presented in a defined sequence; this fits teaching concepts across many disciplines in science and engineering; some extensions to this model based on *run-time decision making* further expands its applicability to more *non-linear* teaching models.

2. Designing content using our system is highly interactive and centered on ease of use, using *graphical interface widgets* that most domain experts and instructors are comfortable with and use on a daily basis; thus, an important goal of this design is to support instructors that are not necessarily trained in software development, which is a major hurdle to extending existing systems to new disciplines. Another goal of this design is to facilitate interactive and exploratory learning by students through the manipulation of data in real time. The graphical user interface provides easily used means for achieving this goal.

3. A key aspect of our proposed design is the ability to *simultaneously present concepts and their associated visual representation*. This provides a powerful means to relate (or provide context to) fundamental concepts with an easy to understand visual representation. We believe this leads to a more active student learning scenario and a better platform for educators, especially for more complicated concepts, such as recursion (in computer science).

Our primary objective of this learning engine is twofold: to empower instructors to impact student learning, and to provide a means for engaging students in interactive learning through exploration (what-if scenarios). In addition, the system provides sufficient flexibility to fit varying teaching styles, in contrast to a one-size-fits-all type system. Thus, two instructors teaching similar courses might have varying styles, and a well designed learning engine should accommodate both of them. A good analogy is a computer game engine used by game content creators, that result in multiple games using the same engine.

We demonstrate the initial design of our visual learning engine in two disciplines, (1) Computer Science : sorting algorithms and recursion, and (2) Electrical Engineering : signal manipulation.

2. RELATED WORK

The majority of work that is relevant to our work pertains to (1) algorithm animation and software visualization[20], and (2) experimental studies that assess their impact on student learning.

Algorithm Animation: Algorithm animation systems came into their own in the eighties with the availability of desktop workstations and bitmapped displays, and the increasing use of *computer animation.* A number of algorithm animation systems were developed, including BALSA [5](Brown Algorithm Simulator and Animator) and its successor, BALSA II[4, 1, 2], TANGO[17, 18] (Transition- based Animation Generation) and XTANGO[19] animation systems. Significant effort via scripting or explicit programming was required to prepare an algorithm animation in many of these systems. Zeus[3], a followup to Balsa II, had some interesting features; in particular, *code and data views could be connected.*

ANIMAL[16] is a more recent system that uses a graphical user interface to build and generate animations, and links animations to text (eg. algorithm pseudocode). While this has primarily been used in computer science courses, its interactive design capabilities make it easy enough to use in other domains. However, ability for user data input and expression evaluation is not possible, and thus, animations are limited to custom designed examples.

JAWAA [13, 14, 15] has features similar to ANIMAL and uses scripting languages to design animations, but does not support user data input or expression evaluation. Jeliot[8] and AlgAE[21] are program visualization systems and automatically generate animations from variables, function calls, and other operations, but the animations are not the most intuitive.

Impact on Student Learning: A number of experimental studies on learning have been performed by researchers to formally evaluate the impact of audio-visual materials, including the use of animations. Several key benefits have been observed, such as (1) combining animation with prediction[6], (2) sophisticated combinations of instructional materials[11], (3) a direct correlation between student learning and student engagement [7]. On the other hand, the meta study by Hundhausen et al.[10] that classified and compared 22 previous studies concludes with a mixed result in the use of algorithm visualization(AV); they report that roughly half of the experimental studies they reviewed had some significant results, while for a roughly equal half "no statistically significant differences could be found between the perfor-
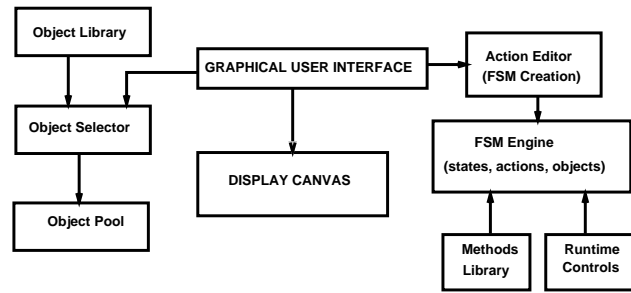

Figure 1: System Architecture

mance of (a) groups using some configuration of AV technology, and (b) groups using either an alternative configuration of AV technology, or no AV technology at all".

In summary, we note that measuring the impact of student learning is complicated and depends on factors such as usability of system, prior knowledge, quality of animations, visual representation, appropriateness and complexity of problem/algorithm, usage scenario, interaction and user input capabilities. We exploit many of these *established and experimentally validated ideas* from these studies in the design and evaluation of our cross-domain visual learning engine, towards adoption by students and instructors in the classroom.

3. SYSTEM ARCHITECTURE

The high level architecture of our system is illustrated in Fig. 1. Broadly, it consists of 4 parts:

Graphical User Interface (GUI): All of the interaction with the learning engine is done via the GUI, thus facilitating ease of use across application domains by students and instructors alike. For example, to design a computer science module that illustrates the behavior of an algorithm, the algorithm is written as a text file and brought into a widget that displays it; other relevant objects, such as variables and arrays, are instanced with appropriate attributes, and finally the corresponding FSM for the module is created. All this is done interactively through the GUI.

Canvas: The drawing canvas is the area where all of the objects and textual material are displayed, manipulated or animated, and also serves the central area for interaction with the system, including objects, concepts and/or algorithms, as well as user data input.

Object Model: We maintain a library of objects. These can be instantiated as required by the application domain. Each object has an associated set of attributes. For example, the simplest object associated with illustrating various computer science algorithms is a memory cell that contains a value; visual attributes include background and foreground colors for the cell, text font and size, and methods to draw and receive input events. All attributes of an object as well as its behavior is controlled via the GUI.

Finite State Machine Model: Building a new module (for instance, a sorting algorithm presentation) essentially involves constructing its finite state machine representation. This involves creating appropriate states and actions to be executed within each state. A special Action object is pro-
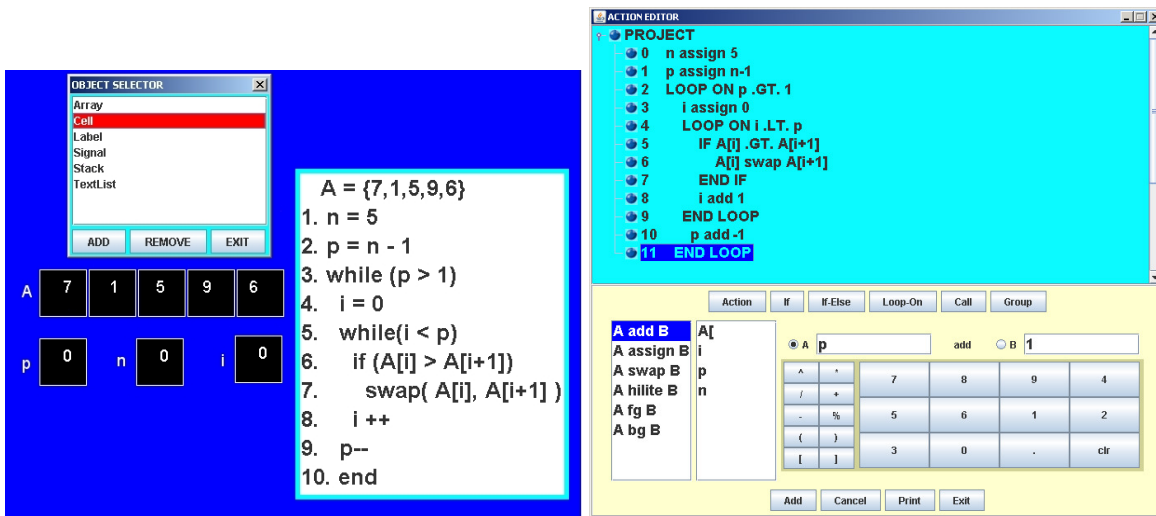
Figure 2: Module Design Procedure:(Left) Array and memory cells instanced and displayed on canvas for bubblesort algorithm, (Right) Action Editor displays the FSM being built(cyan panel), and method dialog box for choosing parameters.

vided to facilitate the execution of actions, and a set of special objects are provided to support branching (similar to if- and if-else statements), repetition (loops), and function calls. Once constructed, a module is *run* by executing its finite state machine representation. Actions are constructed from a Methods Library, which efficiently maintains a single copy of a method to be shared (loosely speaking, objects inherit them) across application domains. The FSM supports all of the needed control flow, including jumps and recursion, in a manner analogous to programming languages. To our knowledge, recursion concepts are not supported by existing systems in such an explicit manner to enable demonstration of recursive algorithms.

4. MODULE DESIGN

We next describe the process of creating and executing a new module. We will use the bubble sort algorithm (in computer science) as an example, as illustrated in Fig. 2.

1. Write the algorithm (as pseudo-code or real code) and save it as a text file. This will be displayed in a widget as shown in the left panel of Fig. 2. The appropriate widget is instanced (via the GUI) using the Object Selector.

2. The Object Selector is used to instance other appropriate objects in the library (an array, and cells corresponding to variables p, n and i are created).

3. After objects have been instanced the Action Editor is used to create the underlying FSM for the module.

4. The Action Editor displays various actions (assignment, addition, attribute changes, etc.) that may be used in a module. Actions are presented in the form 'A method B', where A and B are expressions. For example, an assignment statement is presented as an action of the form 'A assign B'. Selecting a particular action brings up a list of parameters from which expressions appropriate to the method involved may be constructed. When finished, the newly created action is added to the fsm and displayed in the Action Editor.

5. Once the module design is complete the presentation can be run from start to finish without stopping, or it can be run step by step.

5. RESULTS

Our prototype Visual Learning Engine is implemented using Jython[9, 12], that enables Python access to the Java API. Our goal in using Python is three-fold, (1) rapid prototyping with access to extensive capabilities of Java packages, (2) platform independence, facilitating public dissemination, and (3) the ability to perform run-time expression evaluation and decision making. We use Java graphics for all drawing and animation.

We have used our system to construct example modules in two domains, computer science and electrical engineering. Our prototype system includes the following objects: memory cell, label, textlist, 1D array, 1D signal, and graph. The methods library supports operations for assignment, addition, comparison, swap, and signal transformations (scale, translate). Methods such as swap, comparison and signal transformations can be animated to emphasize the mechanics of the underlying concept. With these objects and methods, we can demonstrate a large number of algorithms and data structures, and limited operations on signals.

5.1 Computer Science

We illustrate two important concepts in computer science using our system: sorting algorithms and recursion. These are usually introduced during the freshman/sophomore year in algorithms or data structure courses. Fig. 3(left) illustrates a bubble sort module; an array object holds the data to be sorted, while memory cells (left column) illustrate the variables being used in the algorithm. As the FSM executes, these change value according to the algorithmic step (highlighted in red) on the text widget that contains the algorithm. At this stage of the algorithm, it can be seen that the value 7 is 'bubbling' to the end of the array, with 7 and its neighboring value being swapped. Methods such as swap and assignments are animated to clearly illustrate
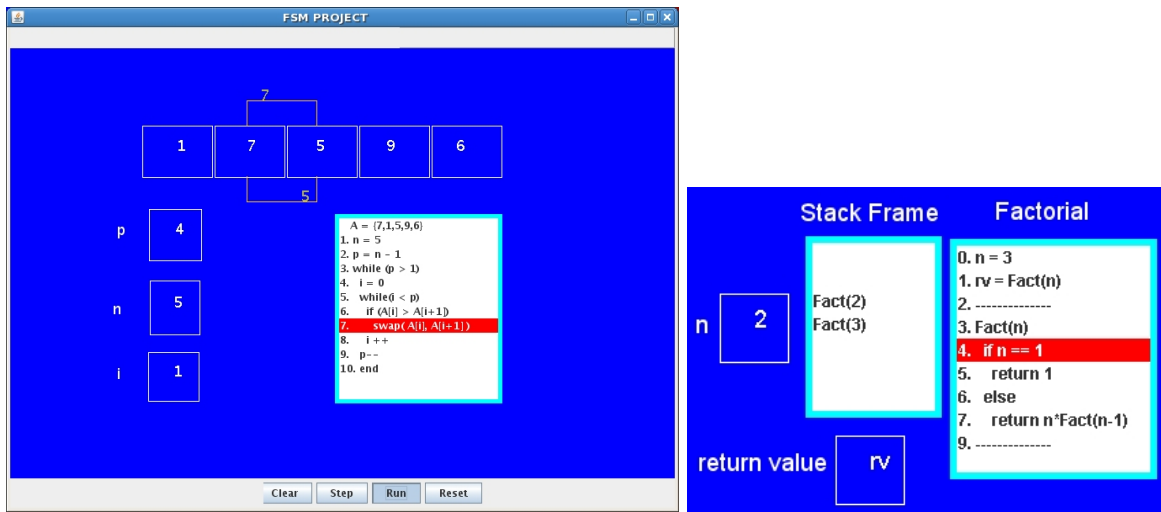
Figure 3: Computer Science: (Left)Bubble Sort, (Right) Recursion - Factorial Example
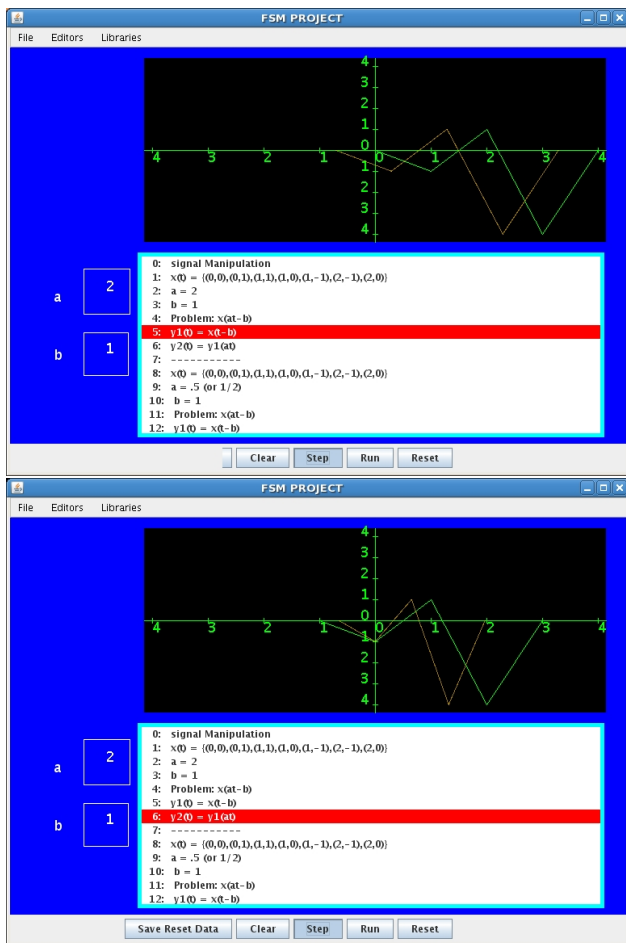

Figure 4: Elect. Engineering Example:Signal Manipulation illustrating $y(t) = x(0.5t - 1)$. (Top) A 1D signal being time shifted by -1.. The green waveform is the original, the waveform in orange animates the transformation to its final position, (Bottom) Signal scaled by 0.5.

the progress of the algorithm, in step with the pseudocode on the right.

The right panel illustrates a factorial computation example, in order to illustrate recursion. We use a stack object to keep the state variables due to function calls. As recursive calls are made, new stack frames are pushed into the stack, and popped on their return.

5.2 Electrical Engineering: Signal Manipulation

A fundamental topic in electrical engineering involves the manipulation of discrete and continuous signals (scaling, time-shifting, time-reversals and combinations of these). Such operations can benefit from a visual representation, in contrast to an *equation/blackboard* only approach. The ability to experiment with different parameters also adds flexibility to make up/try different examples, as well as encourage students to explore further.

Fig. 4 illustrates a 1-dimensional signal manipulation module that was constructed with our system. A signal object was created as a base object, and represented as a discrete set of values. Transformations on the signals manipulate the individual values of the signal. As before, the text widget illustrates the sequence of steps involved in manipulating the signal. Here we show scaling and time-shifting a signal, x(t). The transformation, $y(t) = x(at - b)$ is done by first time-shifting the signal, $(y_1(t) = x(t - b))$, followed by scaling, $y_2(t) = y_1(at)$. As the FSM executes through the sequence, new transformed signals are created at each step. The transformation is animated, so as to show the transition to the final signal form.

6. CONCLUSIONS

We have demonstrated a prototype of a Visual Learning Engine that can form the basis for interactive development of instructional modules by educators across disciplines. A key aspect of our system emphasizes ease of design using graphical user interface widgets, permitting *non-programmers* to use the system; this helps extend its applicability to multiple disciplines. The ability to tightly couple fundamental concepts to visually rich content in a purely interactive design

framework is a unique feature of our system. We believe that this will assist student engagement and learning. A second important aspect of our system is the ability of the user (students, for example) to change input data to test the same algorithm (concept), or explore what-if style scenarios, and as a result promote student engagement and active learning.

We are currently planning user studies for evaluation of our system on qualitative and quantitative basis. In particular, our focus is on (1) ease of use, (2) impact on student learning and (3) cross-domain extensibility.

We have illustrated the use of the learning engine with examples from computer science and electrical engineering. With just a few domain specific objects, it is possible to build modules covering a large number of concepts. For instance, with the array and cell objects, many sorting algorithms can be built (we have built insertion sort as well); the stack object then extends this to recursive algorithms (quick sort, merge sort). Thus, the interactive infrastructure built on an FSM model provides the needed generality and extensibility to other domains.

Future work on the system will be focused on adding multi-media capabilities (images, video, audio objects). This opens up the application of the system to disciplines that focus on multi-media content, or use such content to perform decision making (what-if scenarios, multiple-choice based questionnaires, etc). We also are planning on a more robust object editor that will allow users to modify existing objects and create new objects specific to their discipline. This capability will make this system significantly more powerful and serve as an important resource for instructors and students.

7. REFERENCES

[1] M. Brown. Exploring algorithms using BALSA-II. *IEEE Computer*, 21(5):14–36, 1988.

[2] M. Brown. Perspectives on algorithm animation. In *Proceedings of the ACM SIGCHI '88 Conference on Human Factors in Computing Systems*, pages 33–38, may 1988.

[3] M. Brown. ZEUS: A system for algorithm animation and multi-view editing. In *Proceedings of the 1991 IEEE Workshop on Visual Languages*, pages 4–9, 1991. Kobe, Japan, October 1991.

[4] M. Brown. *Algorithm Animation*. MIT Press, Cambridge, MA, 1998.

[5] M. Brown and R. Sedgewick. A system for algorithm animation. *Proceedings of the 11th annual conference on computer graphics and interactive techniques, SIGGRAPH '84*, 18(3):177–186, July 1984.

[6] M. Byrne, R. Catrambone, and J. Stasko. Evaluating animations as student aids in learning computer algorithms. *Computers and Education*, 33:253–278, 1999.

[7] S. Grissom, M. McNally, and T. Naps. Algorithm visualization in CS education: comparing levels of student engagement. In *SoftVis '03: Proceedings of the 2003 ACM symposium on Software visualization*, pages 87–94, New York, NY, USA, 2003. ACM Press.

[8] J. Haajanen, M. Pesonius, E. Sutinen, J. Tarhio, T. Teriisvirta, and P. Vanninen. Animation of user algorithms on the web. In *IEEE Symposium on Visual Languages 1997*, pages 360–367, 1997.

[9] J. Hugunin. Python and java - the best of both worlds. In *Proceedings of the 6th International Python Conference*, 1997. Oct. 14-17, San Jose, CA. WWW: www.jython.org.

[10] C. Hundhausen and S. D. iand J.T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13:259–290, 2002.

[11] C. Kehoe, J. Stasko, and A. Taylor. Rethinking the evaluation of algorithm animations as learning aids: an observational study. *International Journal of Human-Computer Studies*, 54:265–284, 2001.

[12] S. Pedroni and N. Rappin. *Jython Essentials*. O'Reilly & Associates, Inc., CA, USA, 2002.

[13] W. Pierson and S. Rodger. Web-based animation of data structures using jawaa. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education*, pages 267–271, 1998.

[14] S. Rodger. Introducing computer science through animation and virtual worlds. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, pages 186–190, 2002.

[15] S. Rodger. Using hands-on visualizations to teach computer science from beginning courses to advanced courses. In *Second Program Visualization Workshop*, 2002. Hornstrup Centert, Denmark.

[16] G. Roessling, M. Shuler, and B. Freisleben. The animal algorithm animation tool. In *Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education*, pages 37–40, 2000.

[17] J. Stasko. The path-transition paradigm: A practical methodology for adding animation to program interfaces. *Journal of Visual Languages and Computing*, 1(3):213–236, 1990.

[18] J. Stasko. TANGO: A framework and system for algorithm animation. *IEEE Computer*, 23(9):27–39, 1990.

[19] J. Stasko. Animating algorithms with XTANGO. *SIGACT News*, 23(2):67–71, 1992.

[20] J. Stasko, J. Domingue, M. Brown, and B. P. (Editors). *Software Visualization*. MIT Press, Cambridge, MA, 1998.

[21] S. Zeil. AlgAE: Algorithm animation engine, 1999. Available: www.cs.odu.edu/~zeil/algae.html.