A Visual Learning Engine for Interactive Generation of Instructional Materials

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

K.R. Subramanian  
Computer Science  
The Univ. of North Carolina  
Charlotte, NC 28223, USA  
kers@uncc.edu

Jeffrey Alexander  
Computer Science  
The Univ. of North Carolina  
Charlotte, NC 28223, USA  
jealexan@gmail.com

Drew Linderman  
Computer Science  
The Univ. of North Carolina  
Charlotte, NC 28223, USA  
dtlinder@uncc.edu

Asis Nasipuri  
Elect. and Comp. Engineering  
The Univ. of North Carolina  
Charlotte, NC 28223, USA  
anasipur@uncc.edu

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. 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 flexible, (1) it is based on a finite state machine model (2) instructional modules are generated using graphical interface widgets, facilitating use by non-programmers, and (3) ability to simultaneously present concepts and their visual representation, allowing 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) and Electrical Engineering (signal manipulations).

Methods. Our system is characterized by the following features:

1. All of the module design is performed interactivity via the GUI, including specification of concept definitions and object creation.

2. To provide flexibility, the visualization of a concept is not inherently linked to the concept definition.

3. As the module is being created, a finite state machine is constructed, with the individual steps of the concepts (eg., an algorithm in CS) corresponding to individual states, while the attribute changes (in values, colors, animations) correspond to actions to be executed when the fsm enters that state.

4. Support for control flow, recursion (modeled as state changes).

5. Implemented in Jython to provide platform independence.

Results. We have used our system to construct modules in two domains, computer science and electrical engineering. We illustrate the computer science example in Fig. 1, where both a bubble sort and an insertion sort algorithm are being animated; the yellow panel illustrates the algorithm (pseudocode) while the blue panel illustrates the graphical representation corresponding to a particular step (step 8, highlighted in red). At this step, a swap between the two cells with values of 9 and 6 are in process. This is illustrated by the values being animated (on the canvas) to indicate the swap operation.