Highly Interactive Web-based Courseware

Frank Hanisch Wolfgang Straßer
WSI/GRIS University of Tübingen
Tübingen, Germany
{hanisch,strasser}@gris.uni-tuebingen.de

Abstract
This contribution presents interdisciplinary approaches to develop, interlink, and modify Web-based learning content with special regard to highly interactive content. We describe how we base development of virtual experiments on a toolkit of reusable software components. For proper embedding into courseware we further provide a scripting interface. Authors may create and modify scripts online by using online wizards. Content is organized in databases and rated, annotated, or extended by authors and readers, respectively by the 'community'. We give show cases in the field of Computer Graphics and Visualization.

Keywords
Web-based Teaching, Interaction, Cooperation.

1. INTRODUCTION
Sophisticated Web-based courseware increasingly focuses on active self-learning and therefore includes constructive concepts, namely interactive illustrations [Beall96], or virtual laboratories [Bothun97] containing highly interactive experiments.

While the benefits of interactive self-tests based solely on HTML formulas or click maps are rather low, highly interactive experiments in contrast allow for visual learning through direct manipulation of all essential parameters (mostly bi-directional). However, developing a greater pool of complex experiments is expensive. In [Klein98] we demonstrated how to base development on a toolkit of reusable software components. We present our current Java architecture in section 2.

Unfortunately, actual profit of highly interactive Web-based courseware often is reduced by poorly implemented interlinking of hypermedia elements. In particular, interactive elements appear isolated: they neither can be modified sufficiently (e.g. by choosing parameters or enhancing functionality) nor be interlinked properly with their context (e.g. by synchronizing virtual experiments with a guided tour).

Existing all-in-one solutions like Asymetrix Toolbook or Macromedia Director avoid these deficits but disappoint in creating complex experiments with bi-directional interaction. A more striking approach is to script virtual experiments [Christian00]. We generalized that idea and supplied our virtual experiments’ base component with a scripting interface [Hanisch01]. We discuss details and applications in section 3.

In a further step a community is built and the courseware is enriched with personal aspects. Popular cooperative tools are forum, chat, or annotations [Dietinger98]. The scripting interface may be utilized to implement a multi-user architecture that allows collaborative work with experiments (tutoring or cooperative work [Dix96]). By gathering access statistics and questionnaires the content is evaluated, modified accordingly, and accompanied with a help system [Maurer96]. Also, to guarantee a courseware’s sustainability and its continuous enhancement we have to decentralize factual, technical and didactical knowledge onto multiple authors.

Usually, adequate online wizards allow for distributed modification of the courseware; user input is versioned and, after verification of an editorial board, integrated automatically into the underlying database. A rating system assures quality by maintaining profiles for authors, participants, and content. For now, this task of decentralization has not been solved for highly interactive content. We adapt this approach to scripting and introduce a script database for virtual experiments in section 4.

Section 5 will give show cases of our courseware for Visualization and basics of Computer Graphics.

Figure 1: Basic architecture for virtual experiments.
2. SOFTWARE COMPONENTS

We start with basic concepts of our architecture for virtual experiments (Figure 1). The fundamental idea for creation and modification of virtual experiments is to use a component-based architecture [Klein98].

We separate from objects both their construction (constraints) and their visual information (renderers). Objects are composed to a scene graph that is traversed by actions (picking, dragging, etc.), resulting in a default interaction and visualization for all experiments.

Renderers may hold arbitrary properties to describe their visual appearance (e.g. colors, line strength, strokes, fill type) and implement a property cache to assure efficiency. As users want to select an object’s visual shape instead its geometry, renderers also perform picking. All of our renderers work output-sensitive and render into a standard Java object (Graphics2D). By such an approach an object’s appearance becomes reusable, exchangeable, and efficient.

In contrast to that, constraints represent an object’s constructive information (e.g. orthogonal lines, paths defined by curves). Dependencies are updated automatically and we result in reusable and exchangeable algorithms - which is not only useful for teaching alternative Computer Graphics algorithms. Instead of implementing a constraints solver we simply rely on a data-flow model. We therefore transferred the task to avoid circle definitions that lead to deadlocks to the authors.

For maximum extensibility a plug-in architecture allows for incorporating custom renderers and constraints. After this, our isolated geometry objects (point, vector, mesh, etc.) contain only primitive functionality together with a transformation cache. To provide a graphical user interface (GUI), we further synchronize objects with GUI elements automatically through adapters (e.g. a bounded scalar value with a textual field and buttons for lower and upper bound).

We reach advanced interaction and visualization with a scene graph model. A scene graph collects all information of a graphical scene in an hierarchical structure. Scene graph nodes are traversed by arbitrary actions, e.g. rendering, picking, or dragging. Simple models often implement a global pop/push state that reflects the current state during scene graph traversal. For interaction on scene graph objects, we had to replace this state object by a state tree with arbitrary flags that may be attached to its node (e.g. node is selected, allows all actions or only specific actions). Such a state tree can be optimized straightforward by saving shared states of siblings in their parent node.

Scene graph nodes contain one geometry object together with a matching renderer and an interaction subtree. The latter one assembles several basic scene graph nodes to provide the desired interaction behavior (e.g. a circle may be modified by three interactive points on the boundary or by two points for center and radius).

Figure 2: This experiment we use for teaching spline curves basics. Students have to program and connect Bézier segments with C0, C1, or G1 continuity in order to lead an animated racing car to the goal.

Note that our 2D scene graph contains two different types of transform nodes holding an object and a canvas transformation. This corresponds to the camera node and object transformation known from standard 3D scene graphs. Using a scene graph we result in default interaction behavior and visualization for our objects. This reduces technical issues and enables authors to concentrate on an experiment’s didactical value (as demonstrated in Figure 2).

3. SCRIPTING INTERFACE

Until now, virtual experiments appear isolated within a courseware. We therefore supplied our virtual experiments’ base component with a scripting interface. We allow scripts to import user-defined classes, instantiate objects, and call their methods. Scripts can also be bound to GUI elements.

Scripting enables end users to program experiments by textual commands that are embedded within other hypermedia elements. Instead of overloading the experiment’s GUI we transfer scripting instructions into the text passage or illustration.

We may present special cases directly within the experiment, or match an experiment’s setup with the current step within a guided tour. Instead of merely changing parameters, we may exchange constraints and renderers, construct additional scene graph objects, adapt the GUI, or even import self-defined classes.

We give a show case in Figure 3. The current course text describes the definition and basic properties of a B-Spline curve. The author motivates the spline by comparing it against a Lagrange interpolation. The hypertext is enriched with corresponding scripts that modify the virtual...
experiment to visualize both B-Spline and Lagrange interpolation (Figure 3, left). The learner may continue to work with his experiment. Similar, we may compare the spline with a Bézier curve (Figure 3, right).

**Figure 3:** By embedding scripting instructions into hypermedia content authors may illustrate the current topic directly within the user’s experiment.

### 4. THE COMMUNITY

With reusable software components and scripting we may now develop highly interactive courseware with an adequate interlinking between all hypermedia elements. Nevertheless, one other challenge remains. Learning content has to be prepared didactically and with a well-founded theoretical background. To assure a courseware’s quality, we clearly should decentralize factual, technical, and didactical knowledge and incorporate multiple authors.

Therefore, we developed online wizards [DM98] that allow for Web-based modification of the courseware. We based our online wizards upon a well-defined state machine to comfortably manage authorization, default values, undo facility, and preview. After verification of an editorial board the given data is integrated into the courseware’s underlying database. We implemented only plain HTML editing functionality, as generally the author’s content creation tool allows for HTML export.

Other online wizards realize annotations, a rating system, or help. Based on such evaluation we set up profiles for authors, participants, and content, which, in the long-term, helps us to improve our courseware.

Our next step is to address ourselves to highly interactive content. Providing online wizards for basic information and metadata for virtual experiments is trouble-free, but we also want to assist authors in creating or modifying experiments online.

Consequently, two online wizards deal with Web-based authoring of an experiment’s scripts and respective documentation of created script instances (Figure 4).

We classify scripts as settings (that merely modify parameters) or operations (to create new instances or modify the user interface). As for non-interactive content, scripts may be tested online at the preview step. To avoid complex dependencies in the script sequence, we restrict additional defined scripts to work with the initial set of packages only. Authors will eventually have to create a new experiment with an extended set of packages. Also, some scripts will require changes of existing scripts. To guarantee that scripts stay compatible to previous versions we only permit script modifications that match with a set of pre-defined script patterns.

Then, our rating system takes effect: after a warm-up phase we can identify high-quality experiments and scripts, or bugs.

By using such an approach multiple authors may now create and modify virtual experiments just with browser functionality. Even untrained authors can perform simple modifications guided by existing script examples. Scripting might be simplified by visual programming.

New classes will be traditionally hard-coded. Likewise, entire experiments can be hard-coded and, afterwards, parameters can be declared as script instances. This way authors can hide specific internal details, as well as clear out public accessible information.

So far we dealt with supporting multiple authors. Similar, we now turn towards multiple learners. Apart from the evaluation tools mentioned above we just add another online wizard to implement forum functionality. Forum threads may be directly connected to course chapters or virtual experiments.

To overcome a learner’s isolation we further generalized our base experiment’s scripting interface to a network model that realizes collaborative work. We simply implemented a single-tutor-many-listeners model; the tutor role may be handed over. A server delivers scripts and initial parameter changes to all participants. As constraints update dependent parameter client-side, they heavily reduce the amount of script instructions as well as the net-load.

We only allow registered scripts to be performed in multi-user mode. Our next challenge is how to set up such a set of registered scripts automatically by utilizing the rating system. We might for example register a script if it is rated useful, bug-free, and secure.

Tutoring might be used in a classroom scenario or remote examination. However, to be of value for tutoring, an experiment’s setup should be well-elaborated.

**Figure 4:** Online wizards for virtual experiments guide through authorization, multi-lingual description, metadata, scripting, and preview.
5. CASE STUDIES

To illustrate how highly interactive Web-based courseware may enhance traditional teaching methods, we present another show case of our courseware for Visualization [Hanisch02], namely in the field of color vision.

We found that our students have difficulties in understanding the basics of color perception, in particular the different CIE color spaces, and color defects (Figure 5). In fact, it is very hard to imagine the 3D shape of all visible colors (that resembles to a horseshoe), and scientists prefer to work with 2D projections like the one in the upper left. For instance, we may simulate the effect of red, green, or blue color blindness (protopanopes, deuteranopes, tritanopes) by calculating point projections onto a line (upper right). But our students mostly make wrong assumptions if we provide them with such illustrations only.

A first enhancement is to provide proper interaction. Students may then compare different color spaces, modify projection point and line, and try out different color test plates. More essential for understanding is to visualize not only one (fully saturated) projection, but also different luminance layers, then locate specific color values in CIE color space and compare them to RGB color values.

At last, we again use scripting to illustrate statements within the theory directly in the simulation, e.g. if the text says ‘the lines only seem to be parallel’, we scale down the scene to show that the projection center only lies far away.

![Figure 5: Teaching basics of color perception and color defects. Students may simulate how protanopes, deuteranopes, or tritanopes see the world.](image)

6. CONCLUSION

We demonstrated concepts and prototypes for proper embedding of highly interactive elements within Web-based courseware. Our main focuses were twice: first, to overcome a virtual experiments black-box behavior and interlink them with their environment, and second, to reduce authors’ and learners’ isolation and therefore build up a community. Community members may modify online contents, including virtual experiments, and discuss, annotate, or rate it. Our virtual experiments also allow for collaborative work. We benefit from developing a component-based architecture, a scripting interface that we generalized to a network model, and online wizards.

7. ACKNOWLEDGEMENTS

We wish to thank our student researchers Sven Gottwald and Christian Holzer for the hours they spent in developing our online wizards and the scripting interface.

8. REFERENCES


