# A framework for uniformly visualizing and interacting with algorithms in E-Learning

Alexandros Paramythis, Susanne Loidl, Jörg R. Mühlbacher, Michael Sonntag
Institute for Information Processing and Microprocessor Technology (FIM)
Johannes Kepler University (JKU) Linz
Altenbergerstr. 69, A-4040 Linz
Austria
{alpar, loidl, muehlbacher, sonntag}@fim.uni-linz.ac.at

**ABSTRACT**
E-Learning materials contain more and more interactive elements, which are a unique asset compared to conventional learning materials. But because of a usually fragmented development process (several co-existing generations of design, development over a longer time, many contributors, etc.), both presentation and interaction might vary widely between such materials, even within a single course offering. Furthermore, a common didactic model (what information is present, where and how it can be accessed, etc.) is hard to achieve. In this paper we propose a framework intended to address some of these issues by streamlining didactic, organizational and technical aspects of interactive e-learning examples. This framework has already been used and evaluated in the development of several such examples, which have been deployed in a blended-learning setting at the university level. While it is particularly well suited for visualizing various kinds of algorithms, it has also proven applicable for other types of interactive examples.

**KEY WORDS**
E-Learning, interactive algorithm visualization, framework, uniform interaction and didactic models

## 1. Introduction

The development of E-Learning material is costly. It is estimated that good material requires investments from 10 up to 100 times those for a "conventional" course (see e.g. [1] for a detailed assessment of costs for a single course). One reason for this is the need for interactivity: Scanning a book and placing it online is in all but the most trivial cases insufficient for high quality E-Learning courses. Although such an approach fails to take advantage of the new possibilities afforded by the deployment “medium”, like personalization and interactivity, it still constitutes an “entry point” into the creation of E-Learning materials. As a consequence, a pattern often observed, at least at the university level, is that materials are not just created at once, but rather evolve over time: It starts with more conventional material and is expanded and enhanced over time (an example is presented in [2]) through other kinds of media (audio, video) and more / better interactive elements, e.g. applets or flash animations.

However, this evolutionary development process can result in several problems. Firstly, this process typically involves many persons in the design and authoring of materials, which almost invariably results in several different modes of interaction. Even when strict guidelines are established, these can be interpreted differently and are perhaps not always followed exactly. Even more problematically, not all materials will adhere to the same didactic model, contain similar additional information (like general background or help pages) or support the same additional functionality (e.g. printing).

In this paper we present a framework designed to prevent these (and other) problems. Our overarching goals in the development of the framework have been: (a) to instate uniform didactic and interactivity models in interactive E-Learning examples; (b) to support the implementation phase of such examples; (c) to foster and, where possible, enforce adherence to usability standards and guidelines; (d) to facilitate the deployment of examples in both on-line and off-line forms; and, (e) to ensure a high degree of reusability of the implemented examples, beyond the E-Learning context they were originally intended for. From a didactical point of view, the intention has been to capitalize upon the possibility of standardizing best practices, by encapsulating much-used and proven didactical structures into what has been termed ‘software templates’ [3].

The presented framework comprises mainly of a number of Java libraries and a build environment oriented towards concurrent applet- and application- mode deployment. It has already been used for implementing a large number of individual examples, which are in practical use in blended-learning courses [4] our institute offers for its students.

## 2. Discussion of the interactivity model

Advanced learning material, in particular for courses in technical or natural sciences must contain elements – typically in the form of Java applets – which allow inter-

action. This is one basic concept for visualized simulation and explorative learning in general.

The user interface of such applets should be as consistent as possible. Students should focus on the subject itself and obstructions or the "cost" of familiarization with different interaction paradigms should be kept to an absolute minimum. Learners expect (and benefit from) uniformity and consistency in the materials that comprise individual courses (or, for that matter, a series of related courses) [5].

This demand goes beyond recommendations for the spatial arrangement of elements, or common style guides in general. It necessitates uniform facilities for performing common tasks in this context, including: mechanisms for printouts, screen copy facilities, appearance and function of input and visualization components, standard facilities for accessing background information, context-sensitive help, etc.

The presented framework enforces both a consistent look and feel throughout the set of provided examples from the learners' point of view, and a set of Java libraries and prefabricated output objects, which allow easy embedding of examples into a predefined environment. Furthermore, their design and implementation (including constituent materials) is guided by a set of custom specification documents that detail the structures and policies that must be observed during development.

According to our experience the effort required to become familiar with the constraints enforced by the framework are negligible compared to the cumbersome work of adjusting someone else's examples after they have been implemented. Even more, due to the prefabricated classes and the API, the development of interactive software proved to be both considerably easier and faster.

### 2.1 Didactic aspects

As already stated, one of the primary goals of the framework has been to predetermine and enforce a number of didactic decisions. An advantage of this is that the actual implementers of examples need not be educators themselves; basic engineering skills should still suffice to achieve at least reasonable results. Examples of elements with didactic background in the framework include the following (we focus here on the deployment of examples embedded in on-line course materials):

*Preface page* - As the examples are often included directly within the on-line learning material, they should initially be not too distracting (and also not consume a lot of resources, like screen space or memory). Therefore each example is accompanied by a short introductory page with a general outline of the example's goal, background theory and main aspects exemplified. Through this learners can decide whether to start the applet or not. The actual visualization is begun in a new window upon pressing a button. This allows a compact representation within the material (see Figure 1a), while not constraining the size of the visualization (thus making it possible to easily accommodate different display sizes and resolutions).

*Help text* - Each example must contain a help text (see e.g. Figure 1b). This should not only describe the background of the visualization (which is already in the base material, but might be required when running as a stand-alone application), but rather how to handle it, and is therefore separated into two files (usage and theory). This part is easily overlooked by implementers, but highly important for learners. The framework supports the use of HTML in these help pages, making it possible to reuse portions of the learning material, and also easily enhance it with images or animations if desired. Moreover, no special editing / compilation step (compared e.g. to MS help files) is required.

*Common general layout* - The example visualization window is also standardized for learner's usage. At the top is a bar describing the current state of the algorithm, i.e., which percentage has already been completed. As not all algorithms might allow easy deterministic calculation of the total number of steps, individual examples can also opt for an open-ended presentation (which nevertheless still happens in discrete steps). Compared to a description or a predefined animation this stepwise development of the result better explains the actual working of the algorithm: Each step is shown separately as it happens. To ease interaction with complex algorithms, advancing to the next step can also be automated (at user-controlled variable speeds, and with the possibility to pause / continue at any time). This allows students adapting the presentation to their personal preferences, and facilitates both acquiring the big picture through observing general trends (fast advancement), as well as inspecting details in single steps.

Furthermore, as the general layout of the interactive examples is always the same through the framework, students get accustomed to this and can more easily handle them. This is especially useful as the interaction modes are quite trivial at the beginning (simple examples), but the more they advance in the course, the more complex applets grow, but then the handling is already familiar to them.

### 2.2 Organizational implications

Introducing a framework for educational examples also has some organizational implications and influences the development process. As discussed above, the iterative development model results in a high turnover of persons contributing to the material. Providing written guidelines often does not work satisfactorily, as these might be interpreted differently, are not known in details and following them is generally seen as onerous additional work. But if these guidelines come in the clothing of a framework, they are adhered to much more closely: This reduces the work, using libraries is customary (in contrast to written regulations), and the enforcement is strict (with compile- and run-time- checks in place).

Similar thoughts apply to quality assurance. Some missing information can easily be overlooked, but empty placeholders show this lack clearly. Testing also gets easier, as the elements are always in the same place and verification gets routine, instead of having to look for all the different elements in several places and then probably forgetting one of them.

A given framework can also change the development process. As we have pointed out in the introduction, the development of learning material is often an evolutionary process and, typically for universities, benefits from contributions offered by students, either voluntarily or organized within programming labs. One should not underestimate this valuable source. In fact, students, based on their own experience and way of looking at a particular subject, are often a source of innovative and compelling ideas on how aspects of theory can be turned into interactive visualizations. Through the employment of the framework presented it is possible to engage students in the design and implementation of examples without the potentially adverse effect of personal aesthetics, arbitrary design choices and insufficiently advanced programming skills. Instead of receiving various results with better or worse visualization, which cannot really be used in learning material without a major rework, the best ones can now be easily selected and inserted into the teaching material. All that is required is adding any extra elements missing (e.g. the theory part) and it will immediately fit in with all the other preexisting examples.

Similar considerations apply to teachers. Often they are not quite satisfied with the material (content, completeness, presentation, structure, etc.), including examples. Basing them on a framework allows for easier changes to adapt e.g. the actual visualization or the explanations given. This is also important when considering different target groups [6]: Examples for computer science students could be adapted to pupils in lower levels of education by extending the description (e.g. possible without programming, as these parts are ordinary web pages integrated into the applet through the framework) and removing some details. The framework enables them to focus on the pedagogical necessities while ensuring a common presentation and the presence of all the important elements.

### 2.3 Technical considerations

Technical aspects are also important for the interactivity to avoid e.g. too many different modes of interaction. These are much harder to define in guidelines or enforce, as they depend on the subject matter visualized.

One possibility to improve similar rendering is integrating advanced libraries. One example for this is rendering of formulas [7]. This is a quite hard task, so applet creators would either try to skip this, resulting in poorer learning quality because of missing information, or create their own drawing code. The latter would result in different renderings, which are probably also worse in quality. While merely integrating a library does not enforce its use, it is asserted that the presence will be encouragement enough to use it, especially if such use is facilitated through a well thought-out and easy to understand API. Additional libraries are included for the same reason, e.g. for creating graphs and charts.

Interaction is also unified through the framework. Algorithms must be formulated to consist of distinct incremental steps. This requires separating the significant changes from those rather unimportant. So while for many algorithms a single inner loop might be interesting (e.g. sieve of Eratosthenes: the next number to be removed), for some a subdivision (RSA key generation: individual checks and calculation elements) or several runs together (bubblesort: finding the smallest remaining element as a single step) might be didactically more sensible. Again, the framework does not guarantee a good selection, but it enforces the author to give some thought to this very important issue.

Different modes of deployment are also taken care of: The framework supports presenting the visualization both as an applet and as a standalone application from a common code base. Through this it is suited both for online distribution as well as presentation during a presence phase or on a CD. All the necessities for the implementation of this are taken care of by the framework.

Also, the framework obviously reduces the work for creating new interactive examples: Only the actual algorithm and its visualization is required. The general interaction (showing basic information, starting it, control of the separate steps, etc.) is already complete so only specific parameters and additional interaction methods must be implemented. Through this also average programmers or students can create presentable results.

## 3. Developing with the framework

The software part of the framework has been implemented as a set of Java classes and associated libraries. The high-level embodiment of examples in the framework is termed a *feature*. Features incorporate all the code and supporting elements that make up a full example. Typically, a feature consists of the following: help text detailing the use of the interactive portion of the feature and an overview of the related theory; an “input panel”, where users can enter any input data required for the featured algorithm, configure its parameters, etc.; and an “output panel” where visualization and interaction with the algorithm takes place. In the rest of this section, we will provide an outline of the development process for features employing the framework, looking simultaneously at the standard interactive facilities it offers.

The development of features starts with assembling the accompanying materials. These usually include a one-paragraph description, an HTML page (including associated resources such as images), and an optional image that is used alongside the aforementioned one-paragraph description to visually identify the feature.

Implementation commences with the creation of a Java class that binds together the materials listed above, and provides the framework access to them in a structured way. At this stage, the framework has already enough information for the "background" section of the "input window" (Figure 1a). It also automatically recognizes the presence of a theory help file, adding a menu element that allows users to invoke the respective help window (Figure 1c). Neither of these facilities requires any programming on the part of the framework's user.

Implementation then moves on to the creation of the feature's "input panel" (Figure 1a, lower part). Although its contents are apparently dependent on the particular example / algorithm, the framework supports developers through a set of components with a common look and feel (see e.g. the shaded "note" component in Figure 1a), and facilities for their spatial arrangement.

The next step involves implementing the actual algorithm that is the core of the example. Acknowledging

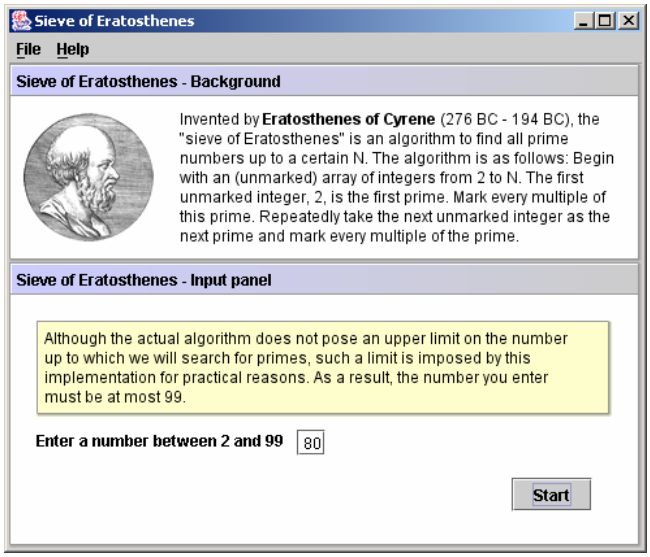


Figure 1a: Introduction / input window

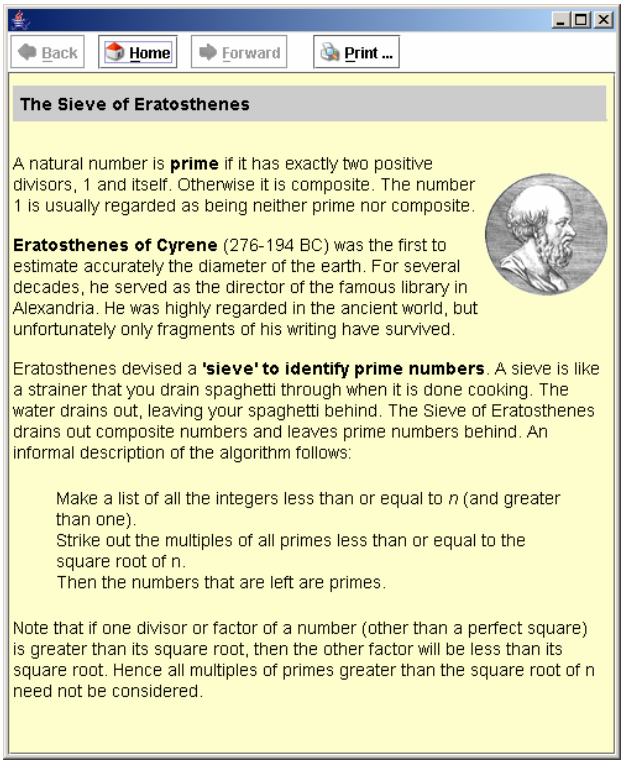


Figure 1c: Help window showing the theory page

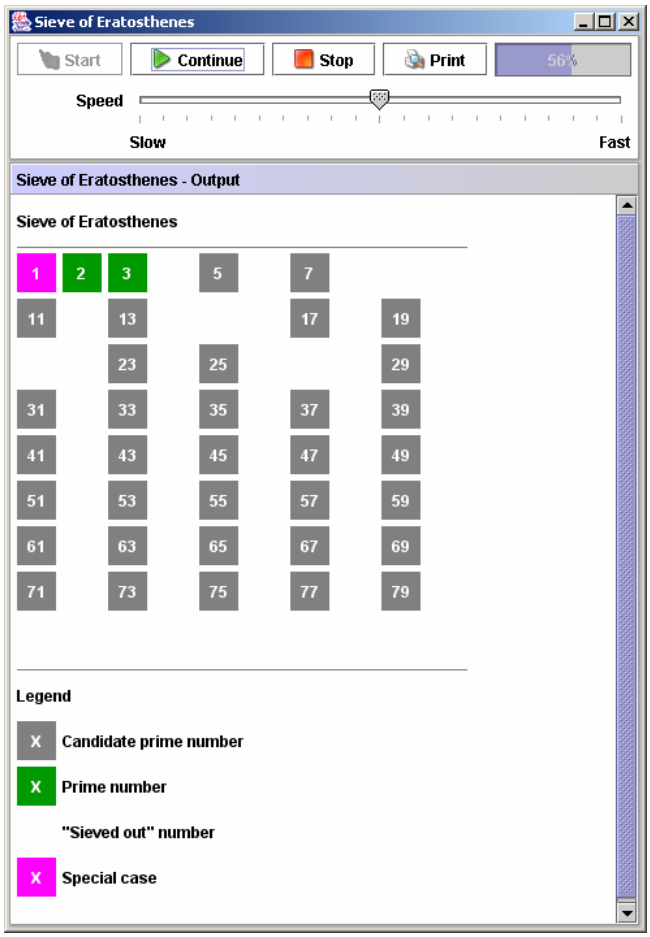


Figure 1b: Visualization / interaction window

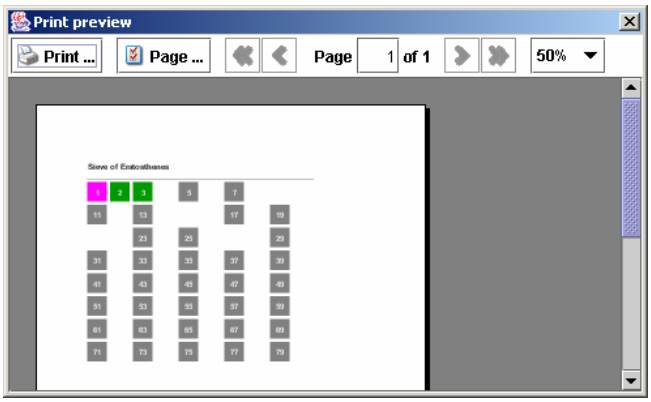


Figure 1d: Print-preview window

**Figure 1:** Standard interactive facilities

the fact that implementation and testing are inherently iterative processes, the framework allows for an incremental approach in this respect. Specifically, initial implementations do not need to expose any details of the individual steps the algorithm involves, or provide actual output. At the other extreme, complete implementations can provide fine-grained information on the total number of steps and the step currently performed, as well as sophisticated and interactive graphical output.

The output window (see Figure 1b) provides an automatically generated section (top part), with components enabling the end user to control the execution / progress of the algorithm. The components available there vary on the basis of the granularity of steps the algorithm implementation exposes, and also supports cases where the total number of steps cannot be calculated, or is too costly to calculate beforehand.

The lower part of the output window, the “output panel” (Figure 1b, lower part) has to be implemented explicitly. To support this task, the framework provides a number of graphics and interaction primitives that can be employed to quickly assemble a graphical output. Furthermore, the framework provides explicit support for the Model-View-Controller (MVC) architectural pattern for interactive software [8]. Following that pattern: each step in the algorithm modifies accordingly a model of the algorithm’s state / progress, the view of the model is automatically updated whenever such changes happen, and the (optional) controller portion can be employed to support direct interaction of the student with elements in the view.

Although the view portion must be implemented explicitly by the developer for run-time presentation, no additional programming is necessary for attaining printable output from that. Specifically, the framework incorporates support for constructing a printer-ready version of the view, including pagination, etc. Furthermore, a print-preview window (see Figure 1d) allows users to inspect output and change printer settings before actual printing.

Once implementation as described above is complete, a custom build system undertakes the generation of archives that can be used for different modes of deployment. In detail, using a simple XML file that drives the process, the build system generates both applet- and application- versions of the feature. Applets are accompanied by an HTML page containing the provided background information and a button that starts the applet. Applications are accompanied by batch / shell files, which make it easy to start individual examples on a variety of platforms. Finally, both the framework and the build systems contain integrated support for internationalization, making concurrent deployment of examples and materials in more than one language possible.

## 4. Practical experiences

The framework was tested through implementing numerous applets, the suitability of which have been validated in several courses. As these are held with the support of the WeLearn web-based learning platform [9], the applet version is used. One important aspect of the platform is however also the offline presentation, so the application form comes in handy e.g. when transferring a course to CD's. The rest of this section outlines some of the example applets that have been implemented with the framework thus far.

"WeLearn.LaVista" is course material demonstrating the concepts of object-oriented thinking and modeling and enabling experimenting with these concepts. Therefore it contains several applets. One example is the "Von-Neumann applet" demonstrating the Von-Neumann cycle (how computer instructions get executed). The students can choose among a number of pre-given short programs, such as a simple calculation, traffic lights, etc., which can then be executed stepwise. Programs vary in complexity, but the operating mode stays the same. Finally, the applet offers the students the opportunity to create further programs on their own, based on a small instruction set.

In the winter term 2004/05, WeLearn.LaVista was integrated into a course provided by the institute for computer science students in their first semester. Feedback showed that the students appreciated experimenting with the applets very much, but that different navigation paradigms (where and how to start a simulation, etc.) sometimes caused confusion. Therefore, it was decided to integrate the applets into the framework, taking advantage of its navigation, documentation, etc. facilities to replace the custom versions in use before. As a result, all WeLearn.LaVista applets now possess the same look and feel and are handled similarly (see Figure 2 for an example). The learners can concentrate on the content and are not confused or distracted by unimportant issues, such as navigation through the applet.

Parallel to the redesign of the WeLearn.LaVista applets, various other applets were implemented for a mathematics course called “IT-Math”. Within this course

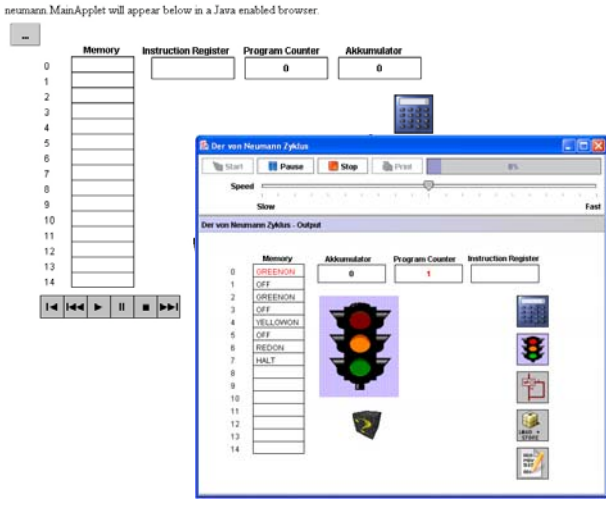


**Figure 2:** "Before" and "after" versions of the "Von-Neumann" applet

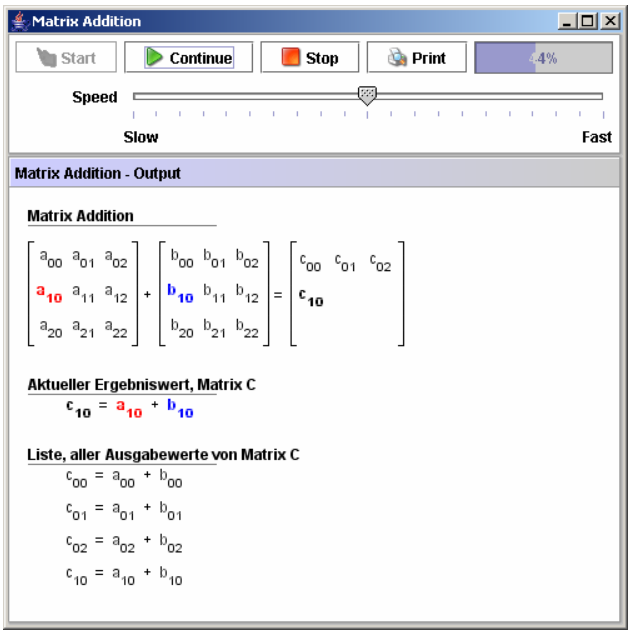


**Figure 3:** "Matrix Addition" example demonstrating the use of the embedded JEuclid library for rendering mathematical expressions

applets demonstrate concepts from graph theory (directed / undirected graphs, adjacency matrix, shortest paths, etc.), number theory, etc. Figure 3 shows the "Matrix Addition" example from this set, which, as its name suggests, demonstrates the steps involved in adding matrices; note that the symbolic version of the addition in this screen-shot is supported through the embedded JEuclid [7] library, which is capable of presenting mathematical expressions in MathML [10]. The "Sieve of Eratosthenes" example in Figure 1 is from this set of features.

## 5. Conclusions

Although the presented approach for unifying interactive and didactic aspects of E-Learning examples has already proven its merits in practice, there are also some drawbacks, the primary one being that the exemplified algorithm must be broken down into individual steps. Whereas this is good from a didactic and interactivity point of view, such implementations of algorithms can differ significantly from the "standard" one, potentially resulting in increased complexity. They are therefore not well suited to teach the implementation of the algorithm itself, but rather for understanding its inner workings.

En route to the further development of the framework, we are currently planning to perform two types of assessment: (a) usability experiments aimed at improving the interactive experience of students using examples created with the framework, and (b) assessment of the framework itself as a development tool.

Improvements that are already under way concern: the integration of facilities to further harmonize presentation and a descriptive definition of required parameters, in this way unifying the interaction aspect even further; the addition of additional deployment methods (currently the framework is oriented towards deployment through applets embedded in web pages and stand-alone applications); and, the development of more "standard" libraries specifically aimed at common knowledge domains (e.g., geometry).

Other possible improvements include support for handicapped persons, including magnification of the visualization (through the framework itself, not the individual example) and special adaptation for handheld devices. Because of their limited screen size special considerations must be taken, e.g. minimization (similar to the magnification mentioned above), support by the framework for scrolling presentation or special modes (e.g. parameters on a separate tab, which is a typical mode of interaction for handhelds, but rather strange for PCs with large screens). Another option we are looking into is support for backtracking, i.e. allowing the user to not only move forward stepwise but also backwards, so the exploration of reasons for certain developments are possible. Although any example can obviously offer this on its own, support by the framework is desirable to also unify this and further reduce work.

## References

[1] N. Hammer, Desing of Design: Die Gestaltung von Mediendesign Lerneinheiten im virtuellen Studiengang Medieninformatik, *Proc. 32nd annual GI conference*, Bonn, 2002, 385-392

[2] B. Gauss, L. Urbas, C. Hausmanns, R. Zerry, Systemic approach to integrating new media into academic teaching – A case study, *Proc. 2003 EDEN Annual Conference*, Eden, 2003, 385-390

[3] W. Jansen, H. van de Hooven, H.P.M. Jägers, W. Steenbakkers, The Added Value of E-learning. *Proc. Informing Science + IT Education Conference (IS2002)*, Cork, Ireland, 2002, 733-746.

[4] S. Loidl, J. R. Mühlbacher, H. Schauer, Preparatory Knowledge: Propaedeutics in Informatics, *Proc. of ISSEP 2005*, to appear

[5] K. Oakes, An Objective View of Learning Objects. *TD (American Society for Training and Development)*, 56 (5), 2002, 103-105.

[6] K. Weicker, N. Weicker, V. Claus, Zielgruppenorientierte E-Learning-Module für das Informatikstudium, *Proc. 32nd annual GI conference*, Bonn, 2002, 90-99

[7] JEuclid: http://sourceforge.net/projects/jeuclid/

[8] A., Glodberg, & D., Robson, *Smalltalk-80: The Interactive Programming Environment* (Reading, MA: Addison-Wesley, 1984).

[9] WeLearn – Web Environment for Learning: http://www.fim.uni-linz.ac.at/research/WeLearn/

[10] MathML - http://www.w3.org/Math/