Adding Intelligent Assessment – A Java Framework for Integrating DGS into Interactive Learning Activities

Andreas Fest

University of Education Schwäbisch Gmünd
Oberbettringer Straße 200
D-73525 Schwäbisch Gmünd

Phone: +49 - 177 - 524 23 53 • Fax: +49 - 7171 - 983-212

fest@cinderella.de

Presented at I2GEO 2010 conference.
Funded by the German Ministry for Education and Research (BMBF).

Abstract

A Java based framework for the development of interactive learning environments based on laboratories containing dynamic geometry applets is presented. DGS applets of different types can be integrated into the same laboratory and interact with each other.

Our framework can be used to enrich interactive DGS constructions and exercises with automated and semi-automated assessment algorithms and allows recordings of learning processes using a capture & replay software.
Two exemplary learning environments based on the framework are presented.

Keywords

congruencies, dynamic geometry, interactive learning activities, semi-automated assessment

MSC 51-04 · MSC 68N99 · MSC 97G40 · MSC 97G50 · MSC 97Q70 · MSC 97Q80 · MSC 97R20

1 Introduction

In common educational settings, Dynamic Geometry Software is applied as a stand-alone application or as interactive applets embedded into HTML documents. In the first case, it is used as a construction tool or in a broader sense as an open environment, which supports geometric explorations. In the second case, DGS applets are used as dynamic construction figures or interactive exercises. A communication between the HTML document and the applet – if at all – is usually implemented via JavaScript and in most cases one-directional from HTML to DGS.

Our aim is to provide the use of DGS as an integral part of more complex learning environments, in order to implement more sophisticated user interfaces for special tasks or adjusted assessment facilities. Especially the implementation of (semi-) automated intelligent assessment algorithms (Bescherer *et al.*, 2010) is an important enhancement of DGS exercises for process-oriented learning.

To enable the possibilities of an integrated assessment a Java based framework for laboratory based learning environments has been developed. In this context an experimental laboratory supports the possibility of learning based on exploration of

special tasks. By using this framework, a collection of laboratories can be embedded in a browser-like user interface.

Applications based on this framework are freely configurable and extendable. We offer predefined laboratory-classes for HTML text content and specific laboratories based on the DGS *Cinderella* (Richter-Gebert and Kortenkamp, 1999). The latter can be customized by user-defined control panels, e.g. HTML forms and configurable toolbars. Additionally, self-implemented control panels and laboratory classes – also for other DGS – can be used.

The presented framework provides multi-directional communication between each component of the application (HTML content, control panels and DGS applets), which uses HTML hyperlinks, scripting modules and the Java event model. The communication model is fully supported by *Cinderella*. Even bidirectional communication between several applets can now be realized.

Recordings of the student's solution process using the capture-&-replay software *Jacareto* (Spannagel *et al*. 2005; Schroeder and Spannagel 2006) are supported.

Our framework is used as a basis for the learning environments *MoveIt!-M* and *Squiggle-M*, developed within the project "SAiL-M – Semi-automated analysis of individual learning processes in mathematics".

The second section deals with the common usage of DGS applets as interactive drawings or exercises embedded into an HTML document. Some of the advantages and limitations of this usage are discussed and an exemplary learning unit using this technology is presented. In section 3 we show how DGS applets can be used to enrich Java desktop applications and present a framework for laboratory based learning environments with embedded DGS applets. In section 4 we discuss how an intelligent assessment can be integrated into our framework. Finally, in section 5 we present the plans for the further development of our framework.

2 Interactive mathematical applets

When Java was published by Sun, one of the most important features claimed was the possibility to create Java applets as small applications that can be embedded into HTML documents and be displayed in some avant-garde web browsers like Netscape (Gosling and Yellin, 1996). First dynamic geometry systems implemented in Java providing the feature of exporting their constructions as Java applets were published in the second half of the 1990 decade. One of the first of them was *Cinderella* (Richter-Gebert and Kortenkamp, 1999), which included an award winning exercise editor featuring a simple automated assessment system.

Since then, the creation of interactive illustrations and worksheets containing dynamic geometry constructions as embedded applets established as one of the most used applications of DGS.

A plenty of web portals containing interactive DGS visualizations and collections of electronic worksheets are published until now, see e.g. (Elschenbroich, 2004), (Richter-Gebert, 2008) or (Intergeo, 2010).

The main advantage of developing mathematical learning units as HTML documents with embedded DGS applets is the simplicity of that task. One just has to create a construction using a DGS and import it into a HTML document, which can be modified

by almost every text editor. Even an adapted user interface can be implemented by using standard HTML controls or simple JavaScript functions.

On the user side the advantage of learning unit published as a collection of interactive HTML documents is that they can used inside a standard web Browser without any necessary installation and that the user always can work with the actual version of the software. An example for such an interactive learning unit was presented by Hoffkamp (2009 and 2010). Also the learning environment *MoveIt!-M* was initially implemented according to this model.

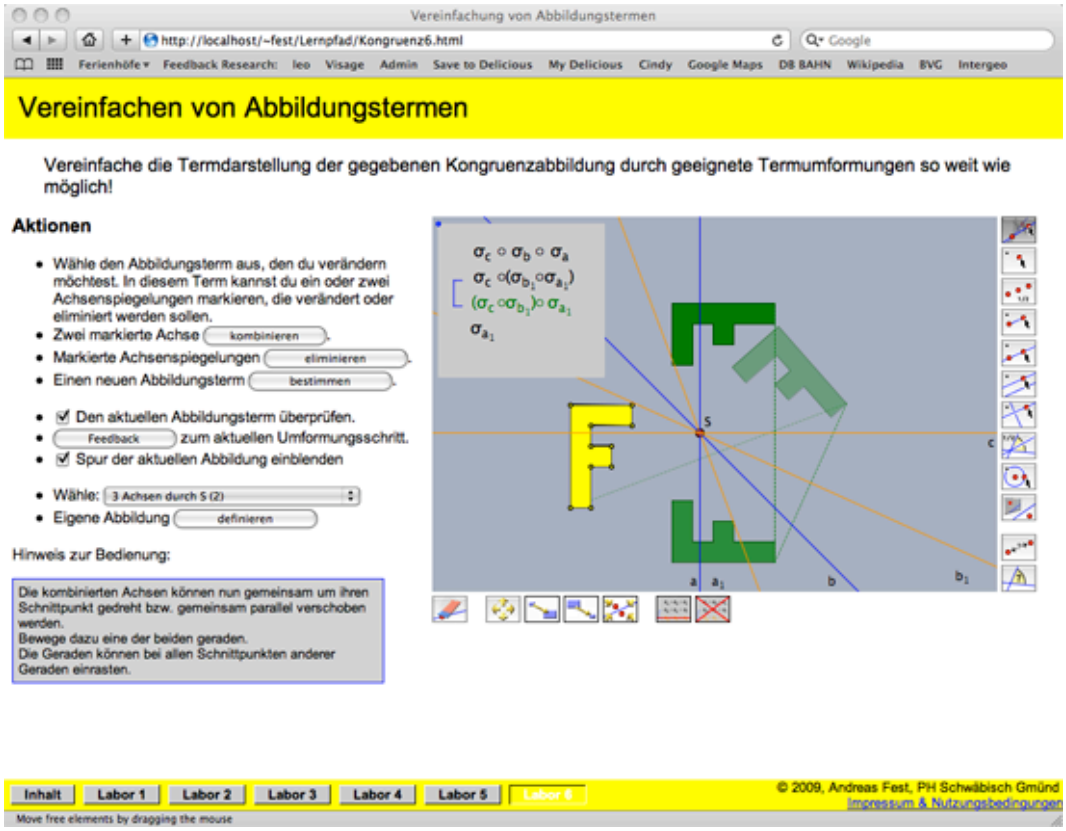
2.1 The learning environment *MoveIt!-M*

MoveIt!-M is a collection of so called learning laboratories, i.e. electronic experimentation worksheets on the topic of geometric congruencies. The laboratories can be used as separate exercises or in sequence as a learning unit.

The basic idea of our collection is the theorem that each congruency transformation can be generated by at most three line reflections. As a consequence, our software opens two different views on congruencies: Congruencies as geometric transformations and congruencies as compositions of line reflections. In the transformation view, main characteristics of a congruency can be explored, e.g. the rotation angle and the center of a rotation. In the composition view, corresponding axes defining a congruency are given (e.g. two lines intersecting at the center of a rotation) and all steps of reflecting a given object along those axes can be displayed.

There are different learning targets that can be followed by the unit. First, the students should get first ideas of and a feeling for congruencies. They should learn how to create different types of congruencies as a composition of line reflections and how to find corresponding axes. Finally they should develop ideas how to reduce a given composition of (more than three) line reflections.

All laboratories are enriched with a wide range of different types of feedback, depending on the special task. The design of some laboratories directs the student to a common standard solution, but all laboratories are also open for differing creative solution strategies.



Picture 1: A laboratory on the geometric reduction theorem for congruencies from the learning environment *MoveIt!-M*.

Picture 1 shows a screenshot from a laboratory about the reduction of compositions of line reflections. A more detailed description of this learning unit is given in (Fest 2010).

2.2 Difficulties & Limitations

During the development of this learning path, we recognized some limitations in the interplay of *Cinderella* applets and surrounding HTML document. The main difficulty is the restricted communication of both parts. Also the HTML document can pass information to the *Cinderella* applet via JavaScript and the *CindyScript* interface of the applet, there is no way to react automatically on changes of the construction inside the applet. Especially, the communication and interchange of data between two different applets inside the same HTML document was not possible.[1]

Further difficulties we had, were:

- the storing and loading user data, for example about the users progress in solving the exercises;
- a complete restart of an exercise after leaving and re-entering a page;
- the system- and browser-dependence of the rendering and the stability.

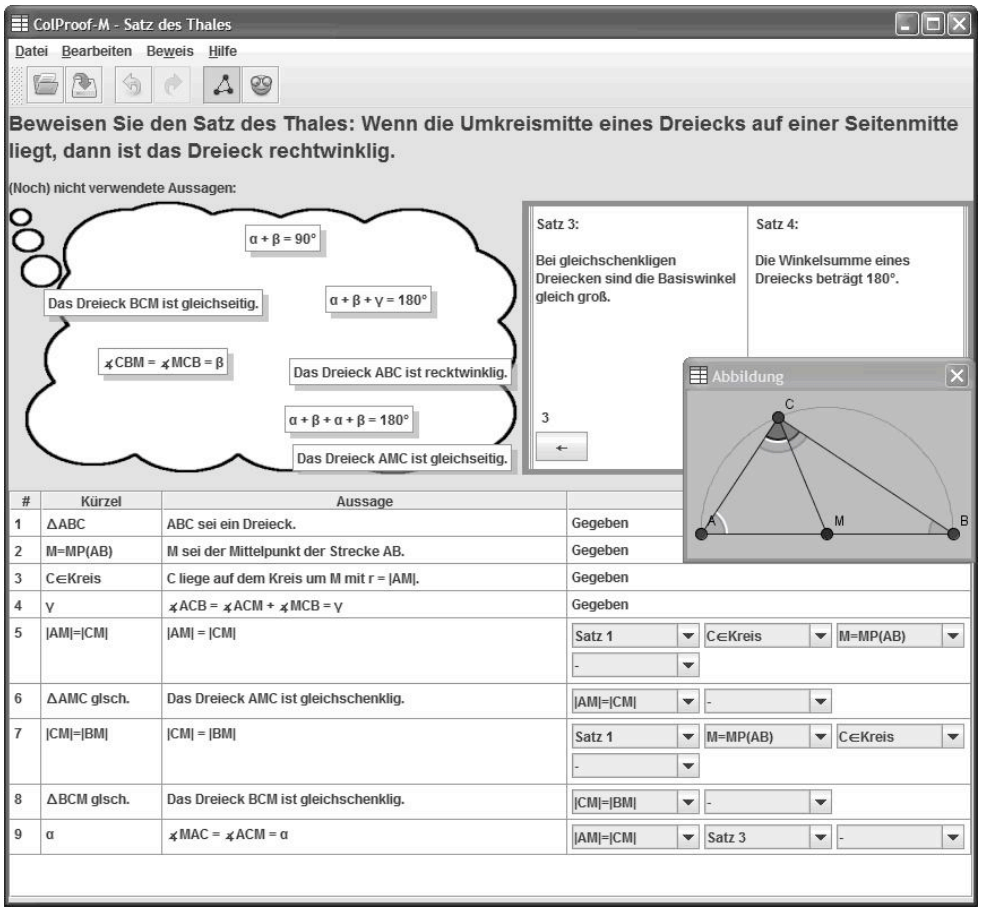
[1] In fact, meanwhile we implemented a possibility to register an event listener for each Cinderella applet. The solution is to load a third applet that implements the `CinderellaEventListener` interface and manages the communication between both *Cinderella* applets.

3 About applets and applications

Usually, Java applications and Java applets are two different ways of implementing user programs, depending on the usage of the software. While Java applications are ordinarily developed as desktop applications that are stored and executed on a local machines, in most cases Java applets are used as web applications, which are executed in a browser environment and are loaded from the internet before starting the program.

Nevertheless, each type of a Java program can be easily transformed into the other one. There are many tutorials dealing with how to transform a Java desktop application into an applet. But the other way round is also possible, and often much easier. First, regard that a Java applet is derived from the Java class `Component` and thus can be embedded into any other Java `Component`, especially into an application `Frame`.

To provide the whole functionality of the applet, the application must instantiate two instances of the interfaces `AppletStub` and `AppletContext`. Both serve as an interface between the applet and its application environment. For example, the `AppletStub` passes the applet parameters to the applet component.



Picture 2: The learning software *ColProof-M* by Herding *et al.* (2010) uses embedded *Cinderella* applets for visualization.

Applying this technology, DGS applets can be used to enrich more complex applications with interactive visualizations. Herding *et al.* (2010) use *Cinderella* applets as interactive proof sketches for their learning software *ColProof-M* on the subject of basic two column proofs in geometry (see Picture 2).

Inspired by this successful reuse of *Cinderella* applets, we decided to convert the learning environment *MoveIt!-M* from a collection of pure HTML documents into a Java application. As a start, the application's user interface was modeled from the existing HTML documents to minimize the implementation effort. Therefore, a customized `HTMLEditorKit` was implemented and a simple scripting mechanism for the handling of hyperlinks was developed.

Starting with this rudimentary application we developed a flexible framework for the implementation of a Java browser for laboratory based learn environments with embedded DGS applets.

3.1 Features of the Laboratory Browser

Laboratory browsers implemented within our framework are freely configurable via property files. Most settings for the appearance and the available content are stored in a default property file and can be overwritten by user-defined files. This opens the possibility for a lecturer to select the available contents of an application and its presentation for his students depending on his lecture or course.

Picture 3: The laboratory on the geometric reduction theorem from the adaption of *MoveIt!-M* to the laboratory browser framework.

Various basic classes to present different types of content are provided. At first, a basic class to display HTML text documents is available. This class uses an Java's default HTML renderer.

A second category of content classes can be used to display laboratories with embedded *Cinderella* applets. Such a laboratory can contain one ore more independent *Cinderella* applets. The user interface of those laboratories can be designed either by common HTML forms, or by labor-specific Java components. Additionally, toolbars containing *Cinderella* toolbar controls and self-defined toolbar buttons can be defined.

Own content classes can extend the available classes or define new types of content, e.g. to display other DGS applets.

We handle hyperlinks inside the HTML document or user interface by an own interpreter that also allows application- and labor-specific scripting commands. For example, *CindyScript* commands can directly be executed by an embedded *Cinderella* applet by activating a hyperlink providing the `href="cdy: cindyscript command;"` tag. Also, hyperlinks to external content that can be opened with the system dependent default application are possible. A typical usage is to link to tutorial video streams uploaded to YouTube. Finally, definitions of terms or notions used inside the content pages can be directly linked to an explanation in an attached glossary.

3.2 Multi directional communication

To provide interactivity of the user interface at a most flexible level, the application must allow bidirectional communication between all of its components. In our framework the communication and data transfer is implemented as follows:

From laboratory to applet

We use the applet specific interfaces for data exchange. Since these interfaces are program specific, we need adapted laboratory classes for each supported DGS implementing its interface.

The DGS *Cinderella* provides two different programming interfaces. We use the *CindySkript* programming interface to control the applet from outside. The applet has two methods `doCindyScript()` and `getValueCindyScript()` available to execute *CindyScript* code fragments. The first method are called by self-defined toolbar buttons, HTML controls and Java components of a laboratory when they are linked with the `href="cdy:"` tag. The second method returns an answer string as result and can also be used by self-defined user interfaces.

Additionally, the common *Cinderella* controls can be added to a toolbar. These controls communicate directly with the kernel of the applet.

From applet to laboratory

For the reverse direction we usually apply the Java event model. The applet must provide an interface to connect event listeners and react on user action by firing an eligible event. The laboratory registers at the applet as an event listener an reacts when it receives the event.

This event model is fully supported by *Cinderella* since the same technology already was used for the processing of semantic events by the capture & replay software *Jacareto* (Spannagel and Kortenkamp, 2009). *Cinderella* fires special events for each important user action. Further construction depending events can be fired either automatically by the integrated theorem proofer or out of *Cinderella*'s programming environment using the `fireevent()` command.

Geogebra (Fuchs and Hohenwarter, 2005) provides a mechanism to send JavaScript commands to the browser environment of a *Geogebra* applet as response on user actions. An interface to handle those JavaScript commands by our laboratory browser is currently in development.

From applet to applet

To establish a communication between different applets, the surrounding laboratory serves as a mediator. The transmitting applet fires an event that is processed by the laboratory. If the laboratory detects that another applet has to react on the event, it sends according commands to the receiving applet.

For the reimplementation of our learning environment *MoveIt!-M* we adapted this mechanism to separate the rendering of the term or description of a congruency from the construction plane (see Picture 3). The software *Squiggle-M* described in section 4.3 uses two independent *Cinderella* applets to display a ladder diagram and the graph of a function side by side (see Picture 4). Changes in the one representation of a function cause immediate changes in the other representation as well.

4 Intelligent Assessment

To support the student's learning process suitable learning software should open the door for individual learning paths to the students (Schulmeister 2007). An intelligent feedback system that offers more information than just a "right" or "wrong" can aid the individual learning process. The availability of feedback can differ in its timing, its presentation, and its information content. Feedback can be given immediately after each user action, delayed, on demand, or after completing a session (Cohen 1985). It can be given visually or acousticly, and in iconic/graphic or textual form. Visual feedback can be presented animated or statically. Park and Gittleman (1992) claim that animated visual feedback can be superior to any static type of feedback.

The information content of feedback can vary between "verification feedback" and "elaboration feedback" (Pridemore and Klein, 1991). While the "verification feedback" just informs about the correctness of a solution, an "elaboration feedback" presents the correct solution and an explanation. Sometimes it may also be sensible to show only partial solutions.

Bescherer *et al.* (2010) discuss the necessity of and the requirements for intelligent assessment in mathematical education. But especially in mathematics there often is an innumerable amount of conceivable solution strategies, and they cannot be validated automatically.

4.1 Semi-automated Assessment

But fortunately this is not necessary in our approach: Most students follow one of a few common solution strategies. Even the mistakes students make are mostly of the same type, as they are based on standard misconceptions. By removing these automatically detectable solutions from the analysis, usually only a few special cases remain that can be handled by manual inspection of the teacher. This concept is called semi-automated assessment (Bescherer *et al.*, 2010) and can be integrated in many learning environment, in particular for mathematics. While the students get individual feedback on standard solutions or mistakes directly from the software, the teacher is notified of those non-standard solutions. As a consequence, the teacher is relieved from the discussion of common standard solutions. He or she gains more time to analyze interesting exceptional solutions and to discuss unusual problems with the students. Also, using the statistical

data about the occurrence of certain standard solutions, he or she can focus on the most common problems, if necessary.

Corresponding feedback should be available any time the student claims that he needs it (*Feedback on demand*, Bescherer and Spannagel, 2009) and should analyze the whole learning process of the student. The analysis of learning processes requires a complete recording of the student's solution process. Especially for the individual analysis by the teacher this is essential. Such a recording can either be done by the learning tool itself or by using a commensurate capture-&-replay software. Spannagel and Kortenkamp (2009) demonstrate how to use the software *Jacareto* for this task. *Jacareto* records any event fired by a Java application. In the replay mode the teacher can follow the student's solution process. Additional assessment algorithms can be developed to analyze and evaluate special semantic events of the application.

The recording should either be stored locally or on a server via an internet connection and should be made available to the teacher whenever the student asks for individual feedback. Herding *et al.* (2010) describe a framework that realizes semi-automated feedback called *Feedback-M* on demand via sending an e-mail to the teacher whenever the student needs more hint then the computer can offer.

4.2 Implementation for MoveIt!-M

In our learning environment *MoveIt!-M* we implemented the assessment according to the principles of the *"feedback on demand"* pattern (Bescherer and Spannagel 2009). The student can ask for feedback by pressing a button. The feedback is given in differently detailed levels. The first level is a visual feedback given directly inside the geometric applet: *how is the observed object effected by my action? Is the image of the object at the requested position?* The visual feedback can be strengthened as an elaborative feedback by coloring the objects green or red according to the correctness of the student's solution.

Additionally, the student can ask for an informative textual feedback to get more information about his solution and hints for an improvement.

For the presentation of the textual feedback we apply the module *Feedback-M*. Using this feedback module, students can additionally ask for a more detailed feedback at any time by sending e-mail to their lecturer or teacher out of the application. On demand, an automatic generated screenshot of the last handled laboratory is attached to the e-mail.

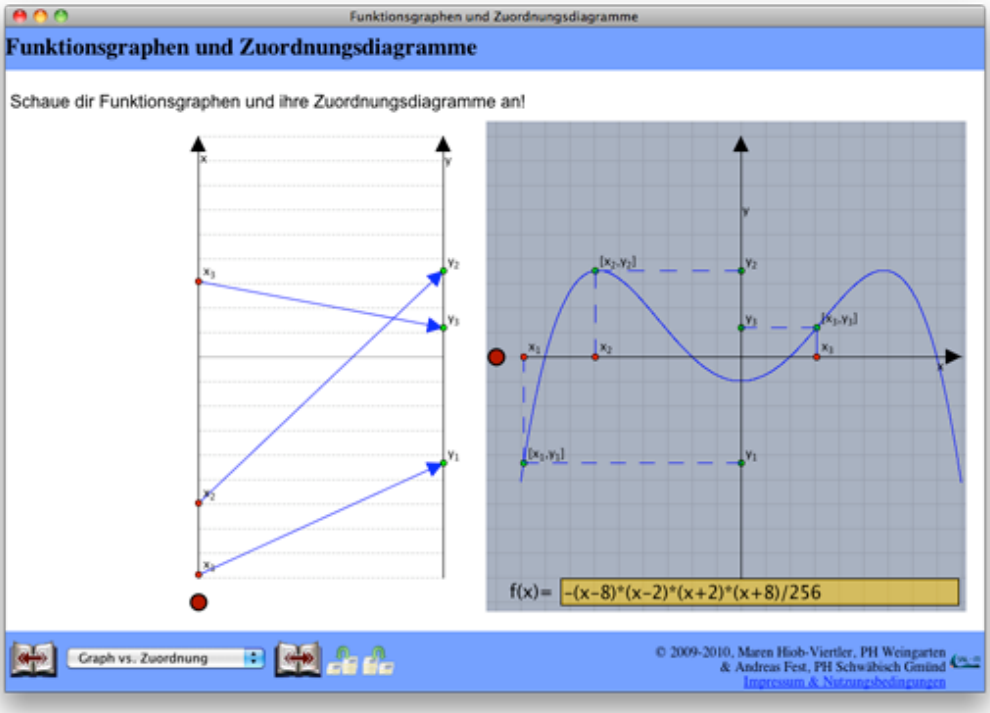
Immediate automated feedback can be realized via the Java event model by analyzing received semantic events. For example we implemented a tutorial on the handling of some geometric objects inside our software. The tutorial consists of a series of small tasks. In each task the student is asked to manipulate the object in a certain way. On every manipulation of the object, the geometry applet sends a semantic event that is received by the tutorial laboratory. When the right action was done, the next task is presented.

Also possible is to start an observer Thread for a laboratory. When a student did not any action that is helpful or necessary for the solution of an exercise, after a while some hints can be shown automatically. When the observer Thread recognizes according semantic events before, those hints are suppressed. Especially, for the introduction of additional user interface elements such automatic hints can lead the student to new ideas.

4.3 Squiggle-M

A second learning environment using our framework is currently in development. *Squiggle-M* (Hiob-Viertler and Fest, 2010, Fest *et al.*, 2010) is an interactive experimentation environment for the conceptualization of the notions of “function”, “injectivity”, “surjectivity”, and “bijection”, using a three-stage approach. In the first stage these notions are initially developed by using finite arrow-diagrams. This focuses on the mapping aspect and the basic idea behind these notions. Within the second stage an extended dynagraph (dynamic ladder diagrams, Goldenberg, 1991) is used to integrate the aspects of change and continuity in the above subconcepts. Stage three links dynagraphs with common graphs of functions visualizing the transition between these representations (see Picture 4).

"Squiggle-M" is a mathematical exploration tool that offers a bundle of experimentation laboratories regarding this purpose. Different representation forms of functions are implemented using the interactive geometry software "Cinderella". The software also presents a collection of open study questions that can be answered within the laboratories by making use of the different representation forms. The individual learning process of the student is reflected by the software's feedback module based on the concepts presented above.



Picture 4: Two representations of functions in *Squiggle-M*.

5 Conclusions & Future Work

Our framework is still in development, but already now it is successfully used to create interactive learning environments with semi-automated feedback mechanisms. It allows data exchange between DGS applets even of different type and without using files in the *Intergeo* file format (Hendriks *et al.*, 2008). The framework can be extended to embed applets from *Cinderella*, *Geogebra*, *GeoNext* or any other Java based DGS.

Our aim is to develop a ready-to-use laboratory browser in which an author doesn't have to write any line of Java code to create interactive mathematical learning activities. Instead he should just have to bundle some DGS constructions, HTML documents and configuration files which were generated by an according authoring tool.

Acknowledgements

This work results from the project "SAiL-M – Semi automated analyses of individual learning processes in mathematics" founded by the German Federal Ministry of Education and Research.

The feedback module *Feedback-M* was developed during the same project at RWTH Aachen. The learning software *Squiggle-M* is also part of the project work and was developed in cooperation with Maren Hiob-Viertler from PH Weingarten.

MoveIt!-M and *Squiggle-M* are available at http://www.sail-m.de/. Our framework will be published under an open source license after a careful revision. To receive a preliminary version, contact the author.

References

Bescherer, C., Kortenkamp, U., Müller, W., and Spannagel, C. (2010). Research in the field of intelligent computer-aided assessment. In McDougall, A., Murnane, J., Jones, A., and Reynolds, N., (editors): *Researching IT in Education : Theory, Practice and Future Directions*. Routledge.

Bescherer, C. & Spannagel, C. (2009): Design Patterns for the Use of Technology in Introductory Mathematics Tutorials. In A. Tatnall & A. Jones (Eds.), *Education and Technology for a Better World* (pp. 427-435). Berlin, Heidelberg, New York: Springer.

Cohen, V.B. (1985). A Reexamination of Feedback in Computer-Based Instruction: Implications for Instructional Design. *Educational Technology*, 25(1), 33-37.

Elschenbroich, H.-J. (2004). Dynamische Geometrie. Online. Available at HTTP: <http://www.Dynamische-geometrie.de > (accessed 20 June 2010).

Fest, A. (2010). Creating interactive User Feedback in DGS using Scripting Interfaces. *Acta Didactica Napocencia* 3:2, p. 79-88.

Fest, A., Hiob-Viertler, M. and Hoffkamp, A. (2010). An interactive learning activity for the formation of the concept of function based on representational transfer. *Proceedings of the CADGME 2010*,

Fuchs, Karl and Hohenwarter, Markus (2005). Combination of Dynamic Geometry, Algebra and Calculus in the Software System GeoGebra. In: *Computer Algebra Systems and Dynamic Geometry Systems in Mathematics Teaching Conference* 2004. Pecs, Hungary

Goldenberg, P. *et al.* (1991): Dynamic representation and the development of an understanding of function. In: Harel, E. (editor): *The concept of Function: Aspects of Epistemology and Pedagogy*, Vol. 25. MAA, Washington

Gosling, J. and Yellin, F. (1996). *The Java(TM) Application Programming Interface, Volume 2: Window Toolkit and Applets*. Addison-Wesley Professional.

Hendriks, M., Kortenkamp, U, Kreis, Y. and Marquès, D. (2008). I2G Intergeo Common File Format v1. Online. Available at HTTP: < http://www.inter2geo.eu/files/D3.3-Common-File-Format-v1.pdf> (accessed 20 June 2010).

Herding, D., Zimmermann, M., Bescherer, C. and Schröder, U. (2010). Entwicklung eines Frameworks für semi-automatisches Feedback zur Unterstützung bei Lernprozessen. Submitted to DELFI 2010 conference, Duisburg.

Hiob-Viertler, M. & Fest, A. (2010): Entwicklung einer mathematischen Experimentierumgebung im Bereich der Zuordnungen und Funktionen. In *Beiträge zum Mathematikunterricht 2010*. Münster: WTM.

Hoffkamp, A. (2009). Enhancing functional thinking using the computer for representational transfer. In: *Proceeding of CERME 6*, Lyon.

Hoffkamp, A. (2010). The use of interactive visualizations to foster the understanding of concepts of calculus – design principles and empirical results. Submitted to I2GEO 2010 Conference.

Intergeo (2010). I2geo Intergeo. Online. Available at HTTP: <http://www.i2geo.net > (accessed 20 June 2010).

Park, O.-C. & Gittleman, S. S. (1992). Selective Use of Animation and Feedback in Computer-base Instruction. *Educational Technology, Research and Development*, 40(4), 27-28.

Pridemore, D. R. & Klein, J. D. (1991). Control of Feedback in Computer-Assisted Instruction. *Educational Technology, Research and Development*, 39(4), 27-32.

Richter-Gebert, J. (2009). Mathe vital. Online. Available at HTTP: <http://www.mathe-vital.de > (accessed 20 June 2010).

Richter-Gebert, J. and Kortenkamp, U. (1999). *The Interactive Geometry Software Cinderella*. Springer-Verlag, Berlin, Heidelberg.

Schroeder, U. and Spannagel, C. (2006). Supporting the active learning process, *International Journal on E-Learning*, 5: 245–264.

Schulmeister, R. (2007). *Grundlagen hypermedialer Lernsysteme.* 4th edition. Oldenbourg, Munich.

Spannagel, C., Gläser-Zikuda, M. and Schroeder, U. (2005). Application of qualitative content analysis in user-program interaction research. Forum Qualitative Sozialforschung / Forum: Qualitative Social Research, 6. Online. Available at HTTP: <http://www.qualitative-research.net/index.php/fqs/article/view/469 > (accessed 20 June 2010).

Spannagel, Christian & Ulrich Kortenkamp (2009): Demonstrating, Guiding, and Analyzing Processes in Dyamic Geometry Systems. In: Bardini, C., C. Fortin, A. Oldknow & D. Vagost (editors): *Proceedings of the 9th International Conference on Technology in Mathematics Teaching* (ICTMT-9), Metz