University of Mannheim/Computer Science
Technical Report TR-2013-001

A Collaborative Multi-Touch UML Design Tool

Michael Magin
Department of Computer Science IV
University of Mannheim, Germany
magin@informatik.uni-mannheim.de

Stephan Kopf
Department of Computer Science IV
University of Mannheim, Germany
kopf@informatik.uni-mannheim.de

ABSTRACT

The design and development of software projects is usually done in teams today. Collaborative systems based on multi-touch walls or large table-top screens could support these highly interactive tasks. We present a novel collaborative design tool which allows several developers to jointly create complex UML (Unified Modeling Language) diagrams. We have developed new algorithms to recognize the gestures drawn by the users, to create the respective elements of the diagram, to adjust the edges between classes, and to improve the layout of the classes automatically. Auxiliary lines provide the user with means to align classes precisely so a more consistent layout is achieved. Export functionality for XML and Java code skeletons completes the application; the UML diagram can thus be used in further steps of the software design process. User evaluations confirm considerable benefits of our proposed system.

Keywords

Advanced interaction systems, collaborative multi-touch interfaces, natural interaction

Categories and Subject Descriptors

H.5.3 [**Group and Organization Interfaces**]: Computer supported cooperative work; H.5.2 [**User Interfaces**]: Input devices and strategies; D.2.2 [**Design Tools and Techniques**]: Computer-aided software engineering

1. INTRODUCTION

Throughout the design phase of a software development process, a number of developers and software architects gather to discuss the structure of the project. While they all have expertise they want to contribute to the project, combining this expertise by collaboratively designing the structure is harder than it should be: The tools currently available to design, *e.g.*, a UML diagram, provide no means to effectively work together as a group of people. This is also caused by the fact that these tools use traditional input devices like mouse or keyboard which do not support collaborative work very well.

The Unified Modeling Language[1] (UML) is a standardized general-purpose modeling language for software engineering. The usual way of designing software is to use a modeling tool, in most cases a tool that generates *UML class diagrams*. Yet, these tools are collaborative only in a limited way. Many of the higher-quality UML diagram editors like Rational Rose[2] support collaboration by allowing users to split up the diagram and delegate parts of the modeling to specific teams. More recent versions even integrate collaboration support for social networks. Still, real *same-space same-time collaboration* cannot be achieved using these tools.

A different approach can be found when designing software without using computers. Whiteboards allow multiple users at the same time to work on the diagram, it is easy to instantly discuss the results and collaboratively design the diagram. But this approach also has a number of major drawbacks: UML diagrams, especially in the design stage, are not static. While they are designed, they are constantly changed, leading to the need to erase parts of the diagrams. Also, once the diagram is completed, it has to be ported into a file to allow further distribution and automatic processing.

We propose a system to overcome the disadvantages by creating a collaborative multi-touch design tool for UML diagrams. *Multi-touch* is a novel interaction technique that enables the manipulation of graphical entities with several fingers at the same time. This technique makes direct interaction and collaboration of multiple users much easier compared to WIMP interfaces (window, icon, menu, pointing device). The main advantage over traditional UML modeling tools is the collaborative approach: Using large multi-touch screens, *multiple users* can work with complex diagrams simultaneously. Even if users are not familiar with UML design tools, they can use *intuitive gestures* to draw their part of the diagram. To facilitate the design process, we have added advanced techniques, *e.g.*, the generation of Java code skeletons from the UML diagram or the automatic layout of the diagram using graph drawing algorithms. The key advantages of such a system, compared to a classical whiteboard, are evident: The tool features a *virtually unlimited canvas* on which the items are drawn, it allows classes to be *moved* and *modified* without the need to re-create the whole class, and it supports panning to move to a new area of the canvas to continue drawing. When moving classes around, the relationship edges are automatically adjusted. In the end, the generated diagram can be *exported* in digital form to allow further distribution and automatic processing.

[1] http://www.omg.org/spec/UML/
[2] http://www.ibm.com/software/rational/

Technical Report TR-2013-001, February 2013,
University of Mannheim/Computer Science.

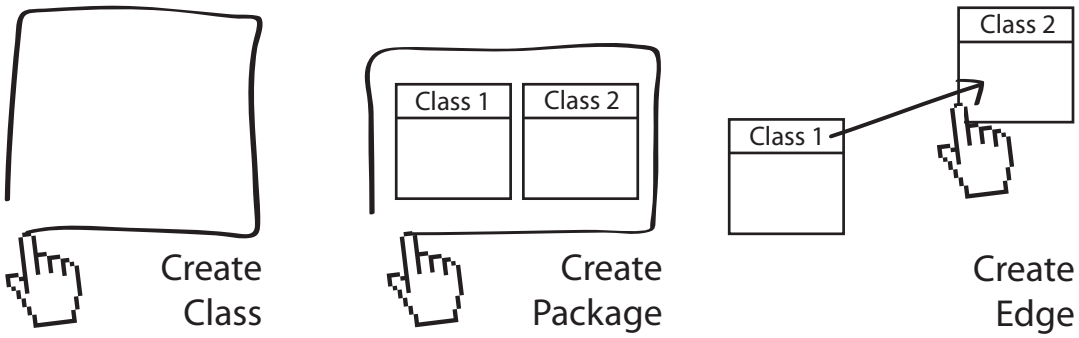

Figure 1: Gestures to modify UML elements (selected)

In this paper, we present a novel system for the generation of UML diagrams. The major goal of our systems is to enable collaborative work on UML diagrams by providing an efficient and user-friendly interface. Although we exemplarily focus on UML, the proposed techniques are generally applicable to the collaborative drawing of diagrams. The major contributions of our approach are:

1. We have developed a complete multi-touch system that enables collaborative work on UML diagrams. Target users vary from professional software developers to groups of undergraduate students who use the system for collaborative learning.
2. Gesture recognition algorithms (see examples in Figure 1) enable an easy interaction and modification of items on the virtually unlimited canvas. The intuitive interaction is especially beneficial for learners.
3. Enhanced functionality facilitates the design process, *e.g.*, Java code skeletons are generated automatically from UML diagrams, auxiliary lines support the drawing, and automatic layout algorithms enhance the visual representation of UML diagrams by rearranging items.

The outline of the paper is as follows: The following section gives an overview of multi-touch technology. Section 3 illustrates the features of our new system. In Section 4, we present user evaluations, and conclude the paper in Section 5.

2. MULTI-TOUCH TECHNOLOGIES

We have developed a complete system for the collaborative design of UML diagrams. To limit the hardware costs, we built a large optical multi-touch table that uses infrared light to capture the input.

Nowadays, most low-cost multi-touch screens use either capacitive, resistive or optical technology. While resistive or capacitive touch-screens are mainly used for smaller devices, many mass-market large screen devices use the following approach: On top of the glass surface of an LCD a plane of infrared light is generated by multiple light emitters. Two low-profile cameras in the bezel of the screen record the interruption of the infrared light created by objects touching the surface. The position of the touch can then be triangulated using information from both cameras. We also use this technique in our system.

In contrast to optical character recognition [7] or classification of arbitrary shapes (e.g., people or gestures) [8, 13], our multi-touch UML tool only uses a small number of different input gestures. This makes it much easier to process the raw input touch data in real-time. There are two main protocols that transmit raw touch data: *TUIO* and *WM_Touch*. *TUIO* is a community protocol created during the development of the Reactable, a multi-touch tabletop that is used as a musical instrument [5]. It is the main input standard supported by all main open-source multi-touch trackers. The TUIO protocol transmits touch events as a tuple of the touch-event identifier and the position of the event on the grid.

With Windows 7, *WM_Touch*[3] (Windows Touch Messages) have been introduced so developers can use the multi-touch functionality integrated in Windows 7. Making use of *WM_ Touch*, we use *MT4j* (Multi-Touch 4 Java)[4] which provides both high-level abstractions and access to low-level functionality. Using inheritance and predefined elements (widgets), it is possible to take advantage of the integrated multi-touch functionality while creating custom widgets. Today, modern multi-touch development frameworks such as MT4j support mostly the same basic set of gestures out of the box. This set consists of the *tap*, *double-tap*, *tap-and-hold*, *drag*, *pan*, *zoom (pinch)* and *rotate* gesture (please refer to [3] for additional information about the gestures).

Our system combines multi-touch input and automatic layout algorithms to support the collaborative work on UML diagrams. Several applications were proposed which use multi-touch displays to simplify the navigation in 2D or 3D environments (*e.g.*, [4, 12]). The technique proposed by Cheng *et al*. [1] is closer related to our system: The authors have developed a multi-touch multimedia system for collaborative learning and testing of students, that focuses on the definition of gestures to interact with educational items. Frisch *et al*. [3] discuss suitable gestures to interact with diagrams, but do not focus on collaborative work.

3. BUILDING A COLLABORATIVE UML DESIGN TOOL

This section describes the design cycle of our collaborative multi-touch system. The visual representation of UML elements and the gestures to manipulate them are presented first. Because not all relevant gestures are supported in current libraries, we had to implement a new gesture recognition algorithm. In the last part, we discuss the specific functionalities that simplify the work with the diagrams by using automatic layout algorithms and methods for

[3]http://msdn.microsoft.com/en-us/library/dd562197/
[4]http://www.mt4j.org/

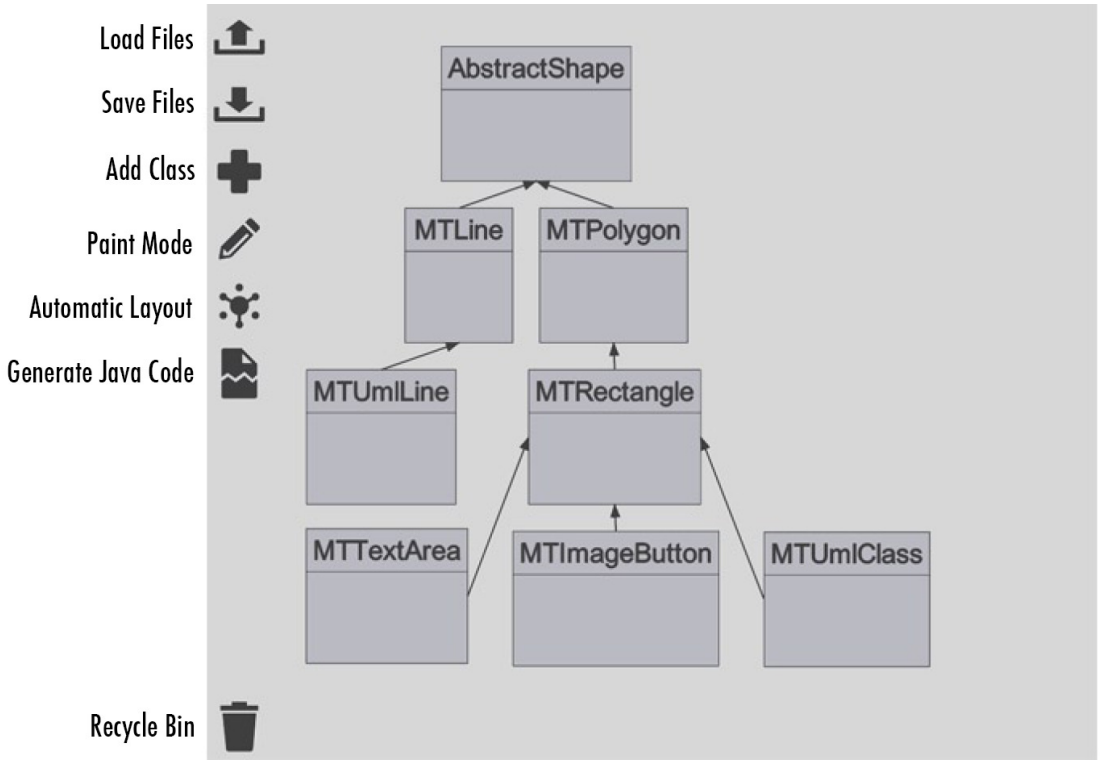

Figure 2: Screenshot of the application (the colors are inverted to enhance the readability of this figure)

data import and export.

3.1 Representation of UML elements

UML class diagrams are a well-known instrument in the software engineering process. They describe the structure of a system by showing the system's classes, their interrelationships, and their attributes. The following elements are supported by our multitouch application: *Classes* are represented as rectangles containing the class name, the attributes, methods, and properties such as an abstract or interface of the class. *Relationships* are represented by different line types and arrowheads. Relationship properties are optional attributes of the relationship and are displayed along the relationship edge. *Packages* are used to group classes, and – as many types of relationships – induce a hierarchical order on the diagram that is used when calculating an optimized layout.

3.2 Usage of Gestures to Manipulate Diagrams

We distinguish between *local functions* that apply changes to selected items on the canvas, and *global functions* which specify the general layout or provide additional functionality like 'loading', 'saving', or 'generating Java code'. Collaborative work is supported for all local functions, whereas only a single user should activate a global method at a particular time. This is the reason why several global functions are only accessible through buttons that are located at a fixed position on the screen.

New *classes* are sketched by drawing a rectangle on the canvas (see Figure 1). *Packages* can be created by either drawing the shape of a package or by drawing a rectangle that encloses one or multiple classes. A double tap on an item opens a new representation, in which the different components can be edited individually. Text is entered using a virtual keyboard which is automatically shown on the screen when a text input field is touched. Creating *edges* is done either by performing a tap-and-hold gesture simultaneously on both classes or by drawing a line from one class to the other. Edges can be edited by using a contextual menu, the type and direction of the edge can easily be selected. Dragging nodes can be done by simply touching the node with one finger and moving the finger. The position of the relationship edges are automatically generated, so they are adjusted whenever a class is moved. Deleting nodes and adjacent edges is done by dragging them to the recycle bin. To delete an edge, the contextual menu of the edge must be used.

The global function *scaling* is not allowed on single nodes for consistency reasons, but can be done on the whole diagram by using the pinch gesture. *Panning* can be done by moving two fingers on the canvas. A set of buttons necessary to call additional global functionality is present at the left side of the screen. The buttons (see Figure 2) allow to save and load the graph using XML files, they trigger the automatic layout algorithm, and the Java code export. A recycle bin placed on the bottom of the left edge completes the overlay menu; elements dragged on top of the recycle bin are deleted from the canvas, including all adjacent edges.

3.3 Gesture Recognition

Many standard gestures are already included in MT4j and can be applied to any component. Yet some of the gestures defined above require a separate gesture recognition algorithm. Therefore, we have implemented a gesture recognition derived from work proposed by Wobbrock *et al*. [15] that consists of four steps: Resample the point path, rotate gesture based on the *indicative angle*, scale and translate, and find the best angle and calculate the optimal score.

The first step, resampling the point path, has to be done as the

algorithm relies on a point-by-point comparison, which requires both the gesture and the pattern template to have the same number of points. It also eliminates the differences that arise when the gesture is drawn at different speeds or different sizes, as the multi-touch system only registers a limited number of points per space and time. The resampled path contains N points that have the same distance from each other.

In the second step, the angle of the path is normalized so that the gesture can be recognized, even if it is drawn at an angle completely different from the template. In consequence, the algorithm is rotation-invariant, which is essential when working with large multi-touch table-tops where users may work and draw from all sides. To normalize the angle of the gesture, the *indicative angle* of the path is used, which is defined as the angle between the centroid of the path and the starting point of the gesture. Note that this normalized angle is merely a point of reference from which further fine-tuning is started.

To account for the different sizes of the gestures, it is scaled non-uniformly to a reference square in step 3. Once this is completed, the points are translated such that the (x, y) coordinate of the centroid of the gesture is $(0, 0)$. In the last step, the average distance d_i between the template T_i and the gesture C is calculated:

$$d_i = \frac{1}{N} \sum_{k=1}^{N} \sqrt{(C[k]_x - T_i[k]_x)^2 + (C[k]_y - T_i[k]_y)^2},$$

where $C[k]$ is the k-th point in the array of points in the gesture. To accurately identify the most probable candidate during the recognition process, the path distance needs to be calculated using the best angular alignment. As we have noted in step 2, the indicative angle is only an approximation of the best alignment. Two angles are constructed at ± 45 degrees of the indicative angle. The angle that generates the less optimal solution is moved towards the other angle until both angles meet.

We have implemented this algorithm in Java, using MT4j's data types and built-in functionality.

3.4 Automatic Layout of UML Diagrams

One of the main advantages of using computers to aid the UML diagram design process is the automatic optimization of the layout of the drawing. Keys to an automatic layout of UML class diagrams are the hierarchical relationships between classes. These relationships are defined either explicitly in the (directed) edges of the diagram or implicitly through the membership of classes in packages. We especially consider the *hierarchy* between classes, *spatial* and *semantic clusters*, the consistent location of parent and child nodes, and *edge crossings*, which should be minimized.

By analyzing basic rules collected from various viewpoints of the UML class diagram design process, Eichelberger [2] identifies several aesthetic criteria. These criteria contain rules imposed by aspects of graph drawing, human computer interaction, software engineering and software visualization.

Based on the idea proposed by Sugiyama et al. [14], we have implemented the following graph algorithm to enable the automatic layout of UML diagrams: In a first step, a directed, acyclic subgraph is generated, and implicit hierarchical information is added as edges to the graph. All nodes are assigned to a specific layer (*rank*), which is defined by the edges. The next step has two main purposes: to reduce edge crossings and to reintegrate edges and vertices that were not hierarchically connected. In the last step, the nodes are assigned their actual positions on the canvas based on their rank.

A further, more visually oriented aid to the diagram layout that was implemented in this work are *auxiliary lines*. They provide a more consistent layout by helping to align classes more precisely. Whenever a class is moved and one of its edges is within two pixels of the horizontal and vertical position of another class, an auxiliary line appears, indicating the position of the other class.

3.5 Data Import and Export

One of the major advantages of a multi-touch based UML modeling tool over a virtual whiteboard is the ability to automatically process the diagram created by the application. Yet to allow the post-processing, the software requires functionality to make the data available to other applications. To simplify data exchange, the system provides input and output based on XML. In addition, the XML files are put under version control. During the design process, unwanted changes can thus be taken back by using an earlier version of the file.

A more specialized approach is the generation of Java code skeletons to provide a jump start to the implementation process by transferring the information contained in the diagram to Java source files. UML classes, attributes, and all their properties are directly mapped to Java. The edges that are least critical to transfer are inheritance and realization edges. Transferring associations is not as trivial: Now, the multiplicities of the edge have to be evaluated to decide how the associations are implemented. Unidirectional associations with a multiplicity of 1 can be implemented by referencing the head class. A multiplicity of *1:n* is implemented by using a single reference on the n side and the data structure *array* on the 1 side. Edges with *n:m* multiplicities now require additional classes to hold the association. Inside these classes, a mapping that relates the n class to the m class in both directions is established.

4. EVALUATION

The objective of the evaluation was to get feedback about the multi-touch interface and about the collaborative support of the system. One of the most important aspects for the learning success is to achieve a high motivation of the students. In previous work, we put our focus on improving learning materials (e.g., lecture recordings [10, 11]). In addition, we analyzed the importance of collaboration for the learning success of students. To improve the activities of the students, we used participatory simulation where users are taking part in the computer-based simulation [6, 9].

We conducted a user evaluation where Information Systems and Business Informatics students familiar with UML class diagrams were asked to perform a test and give feedback about *Advantages*, *Disadvantages* and *Possible Improvements* of the system. In a first step, the users were – without introduction to the system – asked to create a UML diagram from scratch. In a second step, a short explanation of the features of the system and the available gestures was given. Next, users were asked to create a diagram and use all the available functionality. The first observation was that the size of the display is critical for the collaboration: In case of three or more users, large table-top screens are beneficial.

Advantages were seen by the users mostly in the natural way that classes or packages could be created, even without prior instruction (see Figure 3). The students did not have any difficulties with the multi-touch input of UML items. They could create classes, edit them using a double-tap, and add packages by drawing a rectangle around the classes. Once the system and the gestures had been explained, also the creation of relationship edges was seen as intuitive. The possibility to collaboratively work on the same diagram was mentioned as the greatest strength of the system. The students

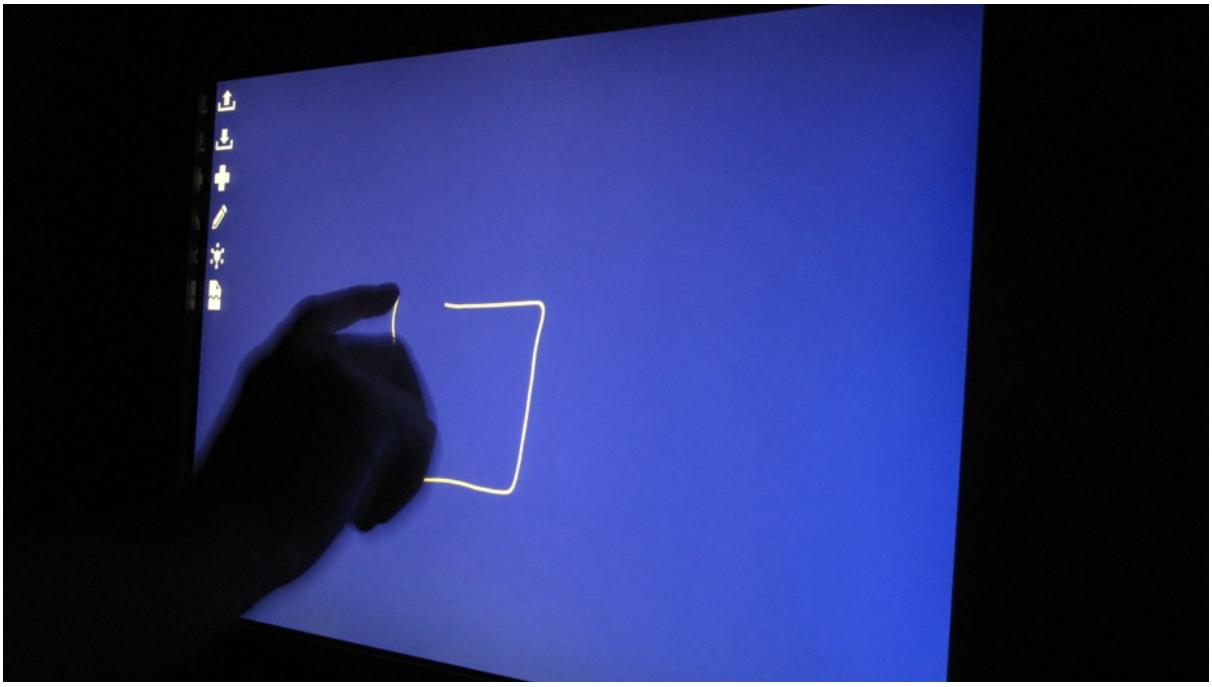

Figure 3: Drawing a UML class on the canvas

had a lot of experience with UML modeling and clearly preferred the multi-touch system to pen and paper. Furthermore, the convenient manipulation of different parts of the application by different people and the intuitive way to define the relationships between those different parts were mentioned to be advantageous. Moreover, the automatic adjustment of the relationship edges even while classes were still being dragged was noted as an advantage, as was the general usage of touch technology to manipulate the diagram.

Disadvantages, at first, were seen in the creation of edges. After instruction, however, users were able to intuitively create and edit edges. More critical was the fact that the collaboration of two or more users may create conflicts. Local functions are not critical, because two users usually do not try to move or edit the same item. Commands that cannot be interpreted by the system are ignored (*e.g.*, if two users try to move the same class in different directions). More problematic are global functions, because scaling or panning might disrupt the work of the other students. Some users found it difficult to use the virtual keyboard to enter text. Furthermore, the lack of contextual help to explain the functionality of the buttons was mentioned as a disadvantage.

As *possible improvement*, context-sensitive help functions were mentioned. Also, a description of the icons in the overlay menu was recommended to ease the first contact with the software. Additionally, the usage of additional input devices, e.g., physical keyboards was asked for, as the virtual keyboard slowed down data input.

5. CONCLUSIONS

We have created a multi-touch based system allowing users to collaboratively design UML class diagrams. The system is made up of independent components and could easily be customized for other applications. People like architects or designers who collaboratively work with diagrams, graphs or drawings might benefit from such a system. The MT4j framework is used to provide basic multi-touch functionality. A gesture recognition algorithm was developed and integrated into the framework to recognize gestures associated with the elements of the UML class diagram. Additional algorithms were developed to enhance the human-computer interaction, like the automatic improvement of the layout of the diagram, auxiliary lines that visualize the current position of other elements in the diagram, export and import functionality for XML files and the generation of Java code skeletons. Users familiar with designing UML class diagrams positively evaluated our system and saw great benefits in both the multi-touch input and the collaborative approach.

While the system already features the aspects mentioned above, a number of aspects still lack an optimal solution. One of the major fields still lacking research is the part of the diagram editing that goes beyond creating elements and moving them on the canvas: The procedure of editing and adding content to the classes. As these activities are mostly text-based, defining this content using a virtual keyboard currently is not intuitive enough. While the recognition of handwritten text, combined with algorithms to semantically analyze the content, could be a solution to the issue, it requires the implementation of complex and very specialized algorithms for handwriting recognition.

To extend the means of collaboration, the inclusion of a merging feature that can merge two diagrams that were edited in different locations also is an idea worth pursuing. It would allow users to concurrently work on the project in different locations, collaborating not only locally, but also remotely.

6. REFERENCES

[1] I. Cheng, D. Michel, A. Argyros, and A. Basu. A HIMI model for collaborative multi-touch multimedia education. In *Proc. of the workshop on Ambient media computing*, pages 3–12, 2009.

[2] H. Eichelberger. *Aesthetics and automatic layout of UML class diagrams*. PhD thesis, University of Würzburg,

Germany, 2005.

[3] M. Frisch, J. Heydekorn, and R. Dachselt. Diagram editing on interactive displays using multi-touch and pen gestures. In *Diagrammatic Representation and Inference*, volume 6170 of *Lecture Notes in Computer Science*, pages 182–196. Springer, 2010.

[4] C.-W. Fu, W.-B. Goh, and J. A. Ng. Multi-touch techniques for exploring large-scale 3D astrophysical simulations. In *Intl. Conf. on Human Factors in Computing Systems (CHI)*, pages 2213–2222, 2010.

[5] S. Jorda, M. Kaltenbrunner, G. Geiger, and R. Bencina. The reactable*. In *Intl. Computer Music Conf.*, pages 579–582, 2005.

[6] S. Kopf and W. Effelsberg. New teaching and learning technologies for interactive lectures. *Advanced Technology for Learning (ATL) Journal*, 4(2):60–67, March 2007.

[7] S. Kopf, T. Haenselmann, and W. Effelsberg. Robust character recognition in low-resolution images and videos. Technical Report TR-05-002, Department for Mathematics and Computer Science, University of Mannheim, Germany, 2005.

[8] S. Kopf, T. Haenselmann, and W. Effelsberg. Shape-based posture and gesture recognition in videos. In *Proceedings of IS&T/SPIE conference on Storage and Retrieval Methods and Applications for Multimedia*, volume 5682, pages 114–124, 2005.

[9] S. Kopf, N. Scheele, L. Winschel, and W. Effelsberg. Improving activity and motivation of students with innovative teaching and learning technologies. In *Proceedings of Methods and Technologies for Learning*, pages 551–556, March 2005.

[10] F. Lampi, S. Kopf, M. Benz, and W. Effelsberg. An automatic cameraman in a lecture recording system. In *Proceedings of the ACM International Workshop on Educational Multimedia and Multimedia Education (EMME)*, pages 11–18, 2007.

[11] F. Lampi, S. Kopf, M. Benz, and W. Effelsberg. A virtual camera team for lecture recording. *IEEE MultiMedia Journal*, 15(3):58–61, September 2008.

[12] J. L. Reisman, P. L. Davidson, and J. Y. Han. A screen-space formulation for 2D and 3D direct manipulation. In *Proc. of the ACM symposium on User interface software and technology*, pages 69–78, 2009.

[13] S. Richter, G. Kuhne, and O. Schuster. Contour-based classification of video objects. In *Proceedings of IS&T/SPIE conference on Storage and Retrieval for Media Databases*, volume 4315, pages 608–618, 2001.

[14] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, Jan 1981.

[15] J. Wobbrock, A. Wilson, and Y. Li. Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *ACM symposium on User interface software and technology*, pages 159–168, 2007.