This is an unedited preprint of Frisch and Dachselt, "Visualizing offscreen elements of node-link diagrams" The final version of this article can be found in Information Visualization, January 2013, doi: 10.1177/1473871612473589.

Visualizing Off-Screen Elements of Node-Link Diagrams

Mathias Frisch, Raimund Dachselt
Interactive Media Lab
Technische Universität Dresden
[mathias.frisch, raimund.dachselt]@tu-dresden.de

ABSTRACT

Visual representations of node-link diagrams are very important for the software development process. In many situations large diagrams have to be explored, whereby diagram elements of interest are often clipped from the viewport and are therefore not visible. Thus, in state-of-the-art modeling tools navigation is accompanied by time consuming panning and zooming. One solution to this problem are off-screen visualization techniques. Usually, they indicate the existence and direction of clipped elements by overlays at the border of the viewport. In this paper we contribute the application of off-screen visualization techniques to the domain of node-link diagrams in general and to UML class diagrams in particular. The basic idea of our approach is to represent off-screen nodes by proxy elements located within an interactive border region around the viewport. The proxies show information of the associated off-screen nodes and can be used to quickly navigate to the respective node. Beyond that, we contribute techniques which preserve the routing of edges during panning and zooming and present strategies to make our approach scalable to large diagrams. We conducted a formative pilot study of our first prototype. Based on the observations made during the evaluation, we came to suggestions how particular techniques should be combined. Finally, we ran a user evaluation to compare our technique with a traditional zoom+pan interface. The results showed that our approach is significantly faster for exploring relationships within diagrams than state-of-the-art interfaces. We also found that the off-screen visualization combined with an additional overview window did not improve the orientation within an unknown diagram. However, an overview should be offered as a cognitive support.

CR Categories: D.2.2 [Software Engineering]: Design Tools and Techniques – *User Interface*; H.5.2 [Information Interfaces and Presentation]: User Interfaces – *Graphical User Interfaces*

General Terms: Design, Human Factors

Keywords: Off-screen visualization, UML, contextual view, interaction, node-link diagrams, navigation

1. INTRODUCTION

Visual representations of node-link diagrams play a very important role in nearly all phases of the software development process. They are used to design the architecture of systems, and they are applied to understand and communicate problems [5]. Over the last 15 years the Unified Modeling Language (UML) [28] has been established as a common standard for designing and modeling software systems. In many situations, UML diagrams can become large with hundreds of nodes and edges. Moreover, within one diagram there can be different elements with a variety of properties. During the design and development process these diagrams have to be explored, created from scratch, and properties have to be added or changed. In many situations these activities are accomplished in a manual way by developers and software designers.

In this work we focus on UML class diagrams as an application example. Class diagrams are most widely applied [7] [39] and feature all the aforementioned characteristics. There are different types of nodes such as classes and interfaces and different types of edges such as associations, generalizations and aggregations. These elements possess a variety of properties such as labels and multiplicities which have to be set or changed.

During the editing process users need to navigate within the diagram. They must be able to focus on a particular node or to move to a certain part of the diagram. Basically, navigation can take place in two ways. On the one hand in familiar diagrams users orient themselves in a "geographic way" similar to map navigation. This means that they know for example the spatial position of a class, the routing of an edge or the direction where a particular class is located. This information is applied to perform navigation. On the other hand knowledge about the diagram topology and structure can be applied for navigation. For example, users often know which nodes are connected, which classes belong to an inheritance hierarchy or which class is at the top of a hierarchy. The actual spatial location of diagram elements or the concrete edge routing is less important in this case. However, contextual information such as properties of relationships (e.g., their types, labels or multiplicities) and types of connected nodes must be available when navigation based on this knowledge should be supported.

Usually, state-of-the-art modeling tools (e.g., [17] [24][40]) only support the map navigation approach. They offer a so-called overview+detail interface. The viewport shows details and is navigated by manual zooming and panning. In addition, a separate window shows the whole information space in miniature to provide an overview [6].

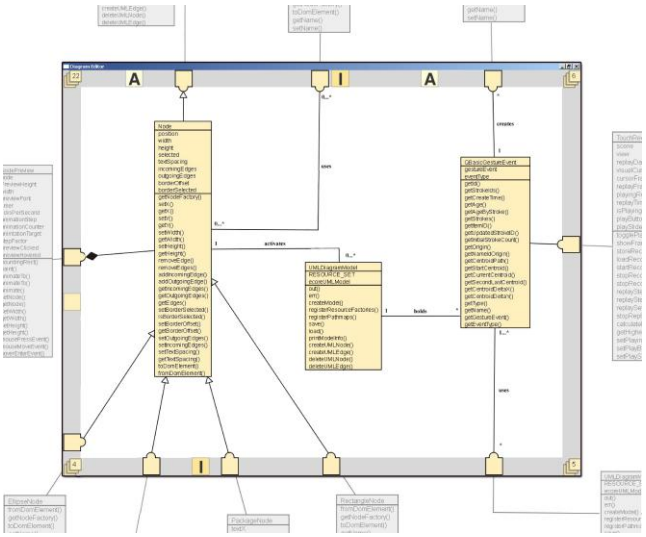

Figure 1. Principle of our solution: The viewport of our UML class diagram editor with off-screen visualization (center). Classes clipped from the viewport (shown outside in gray) are represented by proxy elements located within the interactive border region.

However, for large diagrams the overview visualization becomes very small and unreadable, which is hardly helpful. Beyond that, when zoomed in on a particular element, other elements move off-screen which means they are clipped from the viewport. They are not visible anymore and can only be reached by cumbersome and time consuming panning and zooming. For example, the class diagram depicted in Figure 1 consists of 51 classes. Three particular classes are zoomed in to be able to read their properties, all others are clipped. Thus, important contextual information such as which off-screen nodes are connected with the visible nodes or the types of relationships leading to off-screen nodes is not visible.

To overcome this problem, focus+context techniques have been developed. Overview and detail views are no longer spatially separated, but integrated into a single viewport [6]. Thereby, the content of the focused region is displayed in detail. It is surrounded by the context which is shown less detailed, according to a degree of interest function (DOI) [14]. Often the context information is distorted in a geometric way [35], for example by applying a fisheye visualization. In that way, all information is on-screen. However, distortion can make the access and comprehension of the context difficult. Moreover, traditional focus+context techniques provide little or no information about diagram structures and topologies.

In this paper (which is an extended version of our work presented in [11]) we investigate off-screen visualization techniques for node-link diagrams. Such techniques provide cues concerning elements currently clipped from the viewport. They can be seen as an alternative to traditional overview+detail or distortion oriented focus+context techniques, but can also be combined with them. Up to now, off-screen visualization techniques were mainly applied to mobile devices [1][16]. However, we conceive it as a promising technique to improve diagram navigation as well and extend it for this domain. Our approach offers a zoomable user interface combined with a contextual view displaying off-screen nodes by means of proxy elements. These elements are arranged within an interactive border region of the viewport (see Figure 1). They offer spatial information as well as structural and topological information about elements currently clipped (e.g., the type of clipped nodes or the members of a hierarchy located of-screen). Furthermore, proxy elements serve as links providing automatic navigation to the associated off-screen node. In that way, our technique supports both, map oriented navigation and navigation based on the diagram structure.

In this research we contribute how off-screen visualization techniques can be applied to node-link diagrams in general and to UML class diagrams in particular. We discuss the respective design space of the approach concerning visualization and interaction techniques. More precisely, we contribute techniques which preserve the routing of edges during panning and zooming and present strategies to make our approach scalable for large node-link diagrams. This comprises filtering and clustering of proxy elements not only according to geometric rules but also to rules based on the structure of a diagram. We implemented a prototype for navigating and editing a selected subset of UML class diagrams. This application was used to conduct two user studies. In a formative pilot study we collected observations and comments from participants. These results led to suggestions which concrete techniques of the design space should be combined and were used to improve our prototype. In the second study we compared the performance of the off-screen visualization to a state-of-the-art zoom+pan interface with overview. The results showed that participants performed significantly faster using our technique for given navigation tasks. Furthermore, we found that the off-screen visualization combined with an additional overview window did not improve the orientation within an unknown diagram. However, our results indicate that participants perceived an overview as a beneficial cognitive support.

The paper is structured as follows: Section 2 presents related work. In Section 3 we give an overview of our approach and discuss particular challenges. After that, visualization and interaction techniques are presented in detail in Section 4 and 5. Section 6 describes our prototype for editing and navigating class diagrams. The formative pilot evaluation and the comparative user study are described in Section 7 and Section 8. Finally, we give a conclusion and an outline of future work.

2. RELATED WORK

There are several approaches to support users in navigation tasks for huge information spaces such as node-link diagrams. In general, these approaches comprise zoomable user interfaces, overview+detail and focus+context techniques. A comprehensive overview of these kinds of interfaces is given by Cockburn et al. [6]. In the following sections we will discuss their application to the domain of node-link diagrams.

2.1 Overview+Detail and Zoom+Pan

The overview+detail technique combined with zoom+pan is certainly the most established approach in state-of-the-art diagram modeling tools such as [17][24][40]. Usually an overview is

shown in an interactive separated area at the border of the workspace. It shows the whole diagram in miniature and uses a viewfinder rectangle to indicate which part is currently observed in detail. Users are able to move this viewfinder for panning or can select a certain part of the overview to navigate to this location in the detailed view. There are some approaches which try to improve overview+detail techniques.

In the work of Dwyer et al. [8] a slower but high quality layout algorithm is applied to the detailed view of the currently focused part of the diagram. For the overview a fast but less accurate approach is used. The authors applied their approach also to UML class diagrams and offer semantic zooming [30] by showing different representations of nodes according to the level of detail. In previous work [13], we also investigated semantic zooming techniques for UML. Thereby, we considered nested diagrams. By zooming in a node, a nested diagram becomes visible which describes this node in more detail. Sharp et al. [37] present several techniques to support the interactive exploration of UML sequence diagrams. For instance, different kinds of filters can be applied to the overview of the diagram. The filters result in graying out or culling certain parts. Furthermore, if a particular message is selected in the overview, the detail view shows the source and target object and the respective call stack.

Concerning overview+detail techniques two general problems exist: the overview window occupies additional screen space and some studies indicated that there may be more cognitive load, as users have to switch between both views [6]. Beyond that, Nekrasovski et al. [27] compared zoom+pan to focus+context for a huge tree structure. They applied both conditions with and without overview and found that showing an additional overview window had no influence on the users' performance.

Tominski et al. [45] and Moskovich et al. [25] presented techniques called "Edge-Based Traveling" and "Link Sliding" respectively. They focus on reducing the effort of manually panning for navigating to adjacent nodes in graphs. To achieve that, they apply automatic navigation along edges. With our approach we also support automatic navigation. However, in contrast to Tominski et al. and Moskovich et al. this is possible between arbitrary nodes, not only between connected ones. Furthermore, with our technique no manual mode switch is necessary to get a preview of the target node.

2.2 Focus+Context

In contrast to overview+detail, focus+context techniques integrate both views in one view. Thereby, elements in focus are shown at a high level of detail and those in the context area are condensed according to certain strategies. For example, elements beyond a particular DOI are blended out as in Fisheye Views presented by Furnas [14] or context elements are geometrically distorted [35]. Existing focus+context techniques can be categorized in approaches with global distortion (distortion affects the whole information space) and approaches with local distortion (only some objects of the information space are distorted). Both have been applied to node-link diagrams and graphs.

2.2.1 Global Distortion Techniques

Global geometrical fisheye views have been applied to graphs by Sarkar et al. [35]. The focused node is magnified and all other nodes are geometrically distorted. The authors developed two different approaches to achieve distortion: cartesian and polar mapping. Turetken et al. [46] and Reinhard et al. [32] seize on this approach and apply it to visualize hierarchical nesting of nodes. Particular nodes, e.g. of business process models and data flow diagrams [46], can be expanded to show nested nodes of a finer level. This technique is also applied in ShriMP [49]. Besides fisheye techniques, ShriMP also offers semantic zooming and multi-focus visualization. It has been applied to visualize the structure of ontologies and programs, e.g. by means of call graphs. Jacobs et al. [20] use a fisheye technique in conjunction with UML object diagrams. It serves for visual debugging and dynamically changes the levels of detail of objects according to a DOI function.

Kagdi et al. [23] apply a focus+context approach to classes of inheritance hierarchies in UML class diagrams. In contrast to aforementioned works, they do not use graphical distortion. Instead, context nodes are represented as an onion graph notation.

2.2.2 Local Distortion Techniques

Another way for graph exploration is the application of lenses [2]. Lenses can show additional information [21] or can support graph exploration by local distortion of the layout of the graph. For example, Tominski et al. [44] presented graph lenses such as the *bring neighbors lens*. It can be used to bring connected neighbors of a selected node towards the focused area. Other graph lenses such as the ones presented by Wong et al. [48] or Panagiotidis et al. [29] locally distort the routing of edges to create clutter-free areas. This type of lens was also applied on multitouch enabled displays by Schmidt et al. [36].

Another technique – called *bring & go* – was presented by Moscovich et al. [15]. It moves proxies of adjacent nodes close to the selected node and can be applied in an incremental way (*bring & go* can also be invoked on proxies). Furthermore, Spritzer and Freitas [42] apply a physics-based approach to change the graph layout for exploration. Their prototype allows the placement of magnets which attract nodes with specific attributes.

Tominski et al. [45] developed a radar view mode for graphs. During navigation by means of a pan-wheel, off-screen nodes are projected to the border of the current viewport. This gives the user the possibility to look ahead during panning. In contrast to off-screen visualization, as we propose it in this paper, this technique does not use proxies, does not show off-screen nodes permanently and does not allow interaction with off-screen nodes.

2.3 Cue-based Techniques

Other to the aforementioned approaches, cue-based techniques do not distort or modify the size of elements to realize a focus+context visualization [6]. One option is to show proxies as contextual cues for elements located in the off-screen area. These proxies are often shown as overlays at the border of the viewport. In that way, a contextual view on elements currently clipped is given. In recent years several cue-based off-screen visualization techniques have been developed. They range from arrows (e.g. applied in computer games) to techniques such as Halo [1] or Wedge [16]. The latter were mainly developed for map navigation on small displays of mobile devices. They are designed to indicate parameters such as the existence and the direction of off-screen elements as well as their distance. This is achieved by graphical overlays visualizing the respective parameters. However, in contrast to our proxy-based approach they do not show further

information about the off-screen element such as its type, and they are not interactive.

City Lights [51] is a first sketch for an off-screen visualization approach which uses proxy elements instead of graphical overlays. It realizes contextual views for hypertext systems. For proxy elements different graphical dimensions such as points, lines and 2D objects are discussed. Furthermore, Irani et al. [18] presented Hop, which allows users to navigate to off-screen elements by means of automatic panning. The technique applies a rotating laser beam to create proxy elements near the focused item. An extension is WinHop.[19] It opens an inset which shows the off-screen region around an item represented by a selected proxy element. In this way, the inset serves as a portal into the off-screen area. Recently, an approach similar to WinHop was developed by Ghani et al. [15]. It also applies insets. Thereby, off-screen locations are shown in small separate views arranged along the border of the viewport. The insets provide information about the local neighborhood of off-screen elements and allow panning and zooming. This is similar to the technique developed by Karnick et al. [22]. They applied insets for route visualizations on geographical maps.

User studies on mobile devices were conducted to compare different overlay techniques. The results showed that Halos perform very well but the performance is lowered if the amount of off-screen target increases [3], [34]. A further study by Burigat and Chittaro [4] showed that Wedges are beneficial for more complex spatial tasks, such as ordering off-screen location according to their distance.

The study conducted by Nekrasovski et al. [27] compared zoom+pan with focus+context (a rectangular rubber sheet) for navigation tasks within a large binary tree on a common PC. Halos were used to indicate the position of already visited nodes. Results showed that the zoom+pan interface was faster and demanded less mental effort than the focus+context interface.

These findings encouraged us to apply off-screen visualizations to node-link diagrams. In contrast to Nekrasovski et al., we do not only visualize the geometric location of an off-screen node by graphical overlays. We go beyond this rather simple adaption of already existing approaches and present a technique which applies proxy elements. This can be understood as a combination of focus+context techniques (such as bifocal views [41]) and cue-based approaches. Furthermore, we contribute techniques such as clustering strategies for proxy elements e.g., based on the diagram structure, two different ways of projecting off-screen nodes and visualizing a variety of additional information.

3. THE OFF-SCREEN VISUALIZATION APPROACH

The visualization techniques presented in this paper are based on the off-screen approaches discussed in Section 2.3. We apply them to node-link diagrams in general and to UML class diagrams in particular. This section describes the general idea of our approach and discusses additional challenges which occur when off-screen visualization techniques are applied to the domain of node-link diagrams.

The proposed user interface is structured as follows: The currently focused part of the diagram is shown within a rectangular viewport. This is done in the same way as in common diagram editors. Within this view, navigation takes place by panning and zooming. The viewport is surrounded by an interactive border region (see gray area in Figure 1). It is used to show proxy elements which represent nodes located off-screen.

According to Zellweger et al. [51] there are four different types of information about unseen objects: Awareness, Identification, Navigation and Interaction. We interpret them as requirements and consider them in the following way:

Awareness. The existence of off-screen nodes should be indicated by the visualization technique, so that users are aware of the nodes currently clipped. As mentioned above, we achieve that by applying proxy elements which are displayed within a border region surrounding the viewport. The position of proxy elements is determined by projecting the position of the clipped nodes to the border of the viewport. Different ways of projection are presented in Section 4.1. The edges between off-screen nodes are not visualized within the border region to prevent clutter.

Identification. Commonly, diagrams consist of elements of various types. For example, in UML class diagrams different types of nodes such as classes, abstract classes and interfaces exist. For off-screen nodes the respective proxy elements should allow identifying them in an easy way. Thus, we map existing node types to the color and the labeling of proxy elements (see Section 4.2 and Figure 5 for details). Furthermore, we propose that edges connecting visible nodes and off-screen nodes are attached to the respective proxy elements. This technique ensures that properties such as arrow heads are always visible and the type of the edge can be easily identified. Beyond that, further properties such as edge labels or multiplicities located off-screen are rearranged accordingly to ensure their visibility.

Navigation. With our technique we support manual navigation. The position of a proxy element is dynamically updated during manual panning and zooming according to the position of its associated off-screen node. In that way, the direction of the off-screen nodes is always indicated. The dynamic update is based on the projection mentioned above. In particular, we implemented two algorithms: radial and orthogonal projection (see Section 4.1). Besides manual navigation, we also support automatic navigation. If a proxy is clicked, automatic zooming and panning is started to navigate to the respective off-screen node. This technique allows for a fast and targeted navigation to a clipped node (details can be found in Section 5).

In contrast to approaches such as Halo [1] or Wedge [16], we do not focus on visualizing the distance to an off-screen element. For most of the diagram notations we consider this information as less important compared to other information such as the type of a clipped node or its location in relation to other nodes.

Interaction. Off-screen visualizations should also include interaction techniques. Besides supporting manual and automatic navigation, our proxy elements are interactive and can give further information about associated off-screen nodes on demand, such as previews. These and further interaction techniques are also discussed in Section 5.

Beyond the mentioned requirements, several new challenges have to be taken into account when off-screen techniques are applied to the domain of node-link diagrams. This includes scalability, the shape of proxies and the diagram layout and edge routing:

Scalability. Off-screen visualization techniques usually suffer from cluttered proxies if a large amount of off-screen elements exist. To address this problem we propose automatic clustering and interactive filtering of proxy elements. Different clustering strategies are presented in Section 4.2 and filtering is presented in Section 5.4. A user study (presented in Section 8) showed that our clustering techniques are applicable at least for diagrams with up to 100 nodes. However, the results made us confident that our approach will also support larger diagrams with several hundred nodes. For larger diagrams we propose a specific technique called *Area of Influence* (see Section 4.2.3).

Shape of proxies. Indicators such as arrows, halos or wedges are hard to distinguish from edges and their visual properties (e.g. arrow heads). We decided to apply proxies which resemble the concrete visual syntax of the diagram notation. Therefore, for class diagrams we use proxies with a squared shape (see Section 4.2).

Diagram Layout and Edge Routing. The diagram layout and the routing of edges should be preserved by the visualization technique. For many types of diagrams the layout of nodes and edges can express a special meaning. It is used as a secondary notation [31] and is an important visual guide for users to build a mental map of the diagram. Several layout guidelines for particular types of diagrams exist (to produce aesthetic layouts). For UML class diagrams e.g., within inheritance hierarchies, general classes should be arranged above their subclasses. Further aesthetic rules are presented by Eichelberger et al. [10]. As previously mentioned, edges leading to the off-screen area are attached to proxy elements. This can result in layout changes during panning and zooming. We investigated several solutions for this problem; they are presented in detail in Section 4.1.

4. VISUALIZATION DETAILS

In this section we investigate several design alternatives for all parts of our off-screen visualization technique. We contribute promising solutions and discuss their benefits and drawbacks. We start with issues occurring within the viewport. After that, we discuss the appearance of the proxy elements. Finally, we present different possible designs for the interactive border region.

4.1 Projection

Proxy elements are created within the interactive border region by means of projecting the positions of off-screen nodes to the border of the viewport. Edges between visible nodes and clipped nodes are attached to the respective proxy elements. In that way, the type of the edge is always visible. Furthermore, we suggest applying two border colors for proxy elements. Proxy elements with attached edges have a darker border color than proxies with no edges (see Figure 4). Beyond that, proxies with attached edges are rendered always in the foreground above other proxies and are never aggregated in geometric clusters (see Section 4.2.1). In that way, proxies representing the next connected off-screen neighbors of visible nodes, are easy to perceive and are always directly accessible.

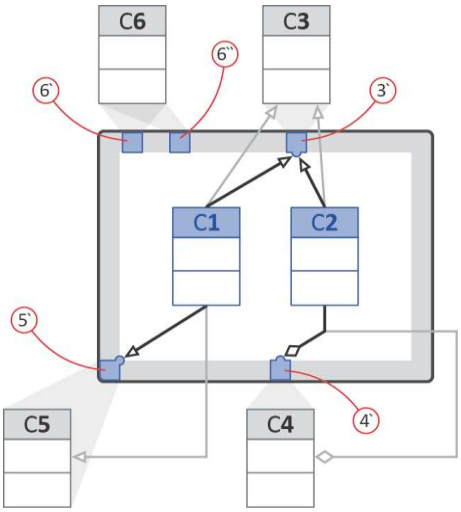

Figure 2. Proxy elements are created by projecting off-screen classes onto the interactive border region (gray area). For edges connected with proxy elements the routing is changed (see aggregation between C2 and C4).

In Figure 2 Class C1 and C2 are both on-screen and connected with Class C3 by generalization relationships. Class C3 is located off-screen and represented by the proxy element 3'. Both generalizations are attached to this proxy element, denoted by the black generalization arrows. Otherwise, the arrow heads would be located off-screen and not be visible for the user (see gray generalization arrows). The edge is automatically released from the proxy element and attached back to the respective node when the node becomes visible due to zooming or panning.

Next, we discuss how projecting nodes in a geometric way affects edge routing and present solutions to make these effects as comprehensible as possible. After that, we present a technique which preserves edge routing completely.

4.1.1 Geometric Projection

The two most obvious and natural ways of projecting nodes onto the border of the viewport are *orthogonal* and *radial* projection. We subsume these two possibilities as geometric projection. For orthogonal projection nodes are projected perpendicular to the border of the viewport. For radial projection nodes are projected towards the center of the viewport. Both ways clearly indicate the direction of an off-screen node. An example for both approaches is shown for Class C6 in Figure 2. Orthogonal projection results in the proxy element 6' and radial projection in proxy element 6''. All other nodes in Figure 2 are projected in the orthogonal way only. A special case occurs if classes are located in one of the four off-screen areas towards the corners of the viewport (see Class C5 in Figure 2). They cannot be projected onto an edge of the viewport by orthogonal projection. Thus, respective proxies are created in the corner of the viewport. Clusters are created if several proxies appear in a corner (for details see Section 4.2).

When geometric projection is applied, the edge routing is changed dynamically during pan and zoom interaction. This happens because edges stick to the proxy elements as described above. In particular, this becomes problematic if an edge is bent and

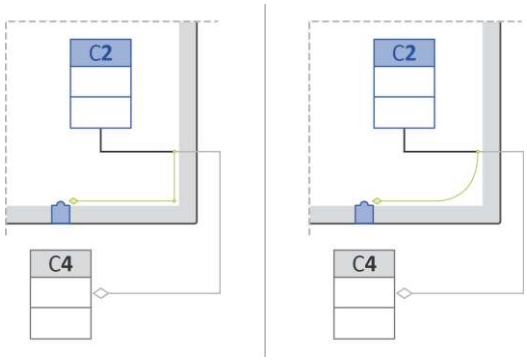

Figure 3. Concept sketch for routing edges along the border of the viewport: rerouting by proxy edges with straight segments (left) and rendered in a rubber band style (right).

consists of several segments. This can be observed in Figure 2 for the generalization between Class C1 and C5 and for the aggregation between Class C2 and C4. The edges are bent and inflection points are located in the off-screen area. In the depicted example a *proxy edge* is inserted from the last on-screen inflection point to the proxy element. This approach does not change the entire edge routing, but still changes the route significantly. We suggest rendering *proxy edges* in a different color than actual edges to signal that they do not represent the original edge (see Figure 2 and Figure 4 where *proxy edges* have a black color).

A permanent change of the edge routing during panning and zooming can be hard to comprehend for the user. Furthermore, guidelines for aesthetic diagram layouts [10] can be violated, as edges crossing each other or edges crossing nodes can occur. In the following we present solutions to make the change of edge routing as comprehensible as possible. A second goal is to preserve at least the routing of the visible part of the edges.To address these problems, we came up with two different solutions: *animated inflection points* and *routing along the border*.

Animated Inflection Points. In order to make the change of edge routing more comprehensible, we suggest animating the inflection points towards the *proxy edge*. The animation starts when the respective node moves off-screen. When the node becomes visible again, the inflection points are animated back to their original position. The drawback of this approach is that even visible parts of an edge are changed. In addition, *proxy edges* can cross other edges or even nodes.

Routing along the Border. Our second solution is to route off-screen edges along the border of the viewport. With this approach the visible part of an edge maintains its routing completely. *Proxy edges* start at the intersection point of the original edge and the border of the viewport and lead to the proxy element (see Figure 3 left). The *proxy edge* is routed according to the original edge (in Figure 3 first downward and then to the left). Another variation of this approach is depicted in Figure 3 right. Here the *proxy edge* is rendered as a smooth curve e.g., by means of a Bezier curve. It can bend dynamically in a rubber band style during panning. This makes the appearance of a *proxy edge* more comprehensible. The general drawback of this solution is that edge clutter can occur along the border of the viewport if an off-screen node has many edges.

4.1.2 Projection along Edges

To avoid the change of edge routing completely, we suggest *along edge projection.* In this approach off-screen nodes which are connected with visible nodes are projected along their edges. In that way, proxy elements appear at the first intersection point of the edge and the border of the viewport. Thereby, the layout of edges is maintained. Figure 4 depicts the same example diagram as in Figure 2 but with *along edge projection*. Off-screen nodes

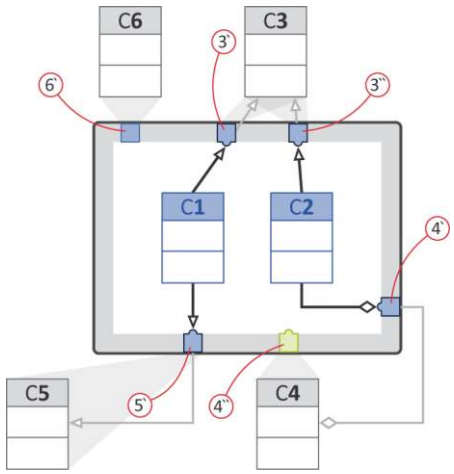

Figure 4. Proxy elements are created by *along edge projection*: proxies appear where an edge crosses the viewport (see proxies 5' and 4'). An off-screen node can be represented by several proxies (see proxies 3' and 3''; both represent class C3). Temporal geometric projection: if proxy 4' is hovered, 4'' appears to indicate the proper direction of class C4.

are projected by means of orthogonal projection if they are not connected with visible nodes. Otherwise they are projected along the edge (e.g., 4' and 5'). There are two further characteristics of this technique. An off-screen node can be represented by more than one proxy element, if the node has several edges. In this case one proxy is created for each edge. This can be observed in Figure 4: for Class C3 a proxy element appears for each generalization

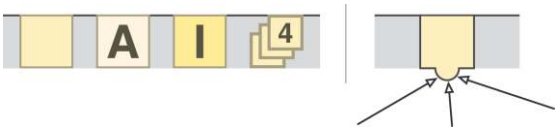

Figure 5. Different shapes for proxy elements (left), from left to right: class, abstract class, interface and a cluster of four nodes. Proxy for a class and attached edges (right).

relationship (3' and 3''). Furthermore, if nodes are connected by means of bent edges the location of the proxy element does not correspond to the off-screen position of the associated node. In Figure 4 the proxy element 4' (representing Class C4) appears at the right border, but the Class C4 is located at the bottom. This can be confusing for the user, as when the proxy element is clicked, the viewport does not move in the expected direction.

We address this problem by applying a temporal geometric projection. It is performed only when a node projected by means of *along edge projection* is hovered with the mouse cursor. The

associated node is additionally projected geometrically. This results in a second proxy element which indicates the actual direction of the node. In Figure 4 the proxy 4'' is a temporal proxy for 4' which appears only when 4' is hovered. However, it has to be clarified if *along edge projection* and temporal projection are comprehensible for the users.

4.2 Proxy Elements and Clustering

In our current implementation we distinguish between four different types of off-screen nodes. For the respective proxy elements we use rectangular shapes with different coloring and labeling. Thereby, the chosen colors comply with the colors of the associated nodes. The applied shapes are depicted in Figure 5 left: proxies for classes are yellow rectangles; proxies for abstract classes are less saturated and additionally labeled with "A" and proxies for interfaces have a higher saturation and are labeled with "I".

Attaching edges to proxy elements can result in clutter. For example, if several edges with arrow heads are attached, the arrow heads can occlude each other. To prevent this problem each proxy owns a so-called edge port. An edge port is a small semicircular extension of a proxy element and provides more space for attaching edges. It has the same color and reaches from the interactive border region into the workspace (see Figure 5 right). Edge ports only appear when the associated off-screen node is connected with visible nodes. We decided that edge ports should be present even if the attached edges have no arrow heads and even if just one edge is attached to the proxy. This clearly visualizes that the edge is attached and makes our visualization more consistent.

To avoid clutter within the interactive border region, we suggest clustering of proxy elements. In that way a scalable technique can be realized for large diagrams. In the following Subsections 4.2.1 and 4.2.2 we present two different ways of clustering proxy elements: geometric and structural clustering. Both can be applied simultaneously and are applicable for diagrams with several hundred nodes. The user evaluation presented in Section 8 shows that the clustering techniques work well with diagrams up to 100 nodes. However, we are confident that these techniques will also be beneficial for diagrams with several hundred nodes. For diagrams with even more nodes we suggest to apply an *area of influence*. Details of this technique are presented in Subsection 4.2.3.

4.2.1 Geometric Clustering

Geometric clustering is applied if several proxy elements are overlapping by more than 30% of their width or height, as they are created at positions very close to each other. In that case, the actual proxies are replaced by a single cluster proxy. For an example see Figure 6 (left hand side), where the classes C3 and C4 are represented by a cluster proxy. They are depicted as an icon which indicates aggregated elements in a stacked way (see Figure 5). Furthermore, cluster proxies show the number of aggregated elements (two in Figure 6). The number is incremented if an associated node moves from the viewport to off-screen and decremented when a respective node becomes visible. Furthermore, for orthogonal projection cluster proxy elements are created for nodes located in the off-screen areas towards the corners of the viewport (see Classes C6 and C7 in Figure 6).

With geometric clustering, proxies are clustered even if there is free space available in the surrounding area. For example, in Figure 6 (left hand side) there is free space above and below the cluster proxy for C3 and C4. For this case, we implemented an algorithm that checks the neighborhood of an existing proxy element. If another proxy element is going to be placed at the same position and free space is available in the immediate vicinity, the proxy element is placed at the free position instead of being hidden in a cluster. Proxies positioned in this way could slightly overlap to indicate that they belong to a cluster.

Whether an *avoid cluster algorithm* is useful depends on the type of diagram. For instance, in state charts or activity diagrams this kind of clustering is certainly not beneficial. For these kinds of diagrams arranging nodes in a vertical or horizontal layout is part

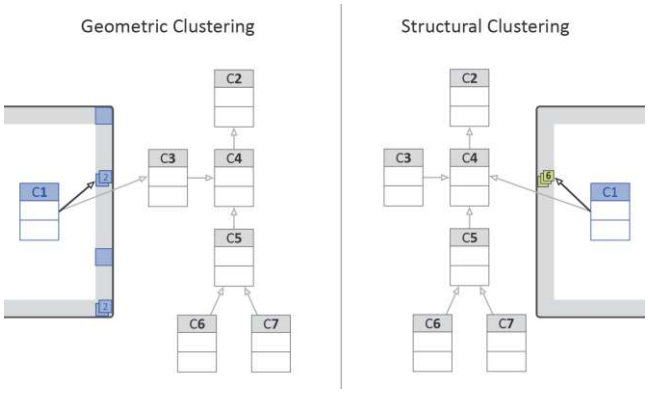

Figure 6. Geometric clustering (left) and structural clustering of an inheritance hierarchy (right).

of the secondary notation [31]. For example, it can be confusing to place proxy elements above each other, although their associated nodes are arranged in a horizontal line.

4.2.2 Structural Clustering

Besides geometric clustering, proxy elements can also be clustered according to structural relationships based on the visual syntax of the particular diagram notation. For UML class diagrams we propose the clustering of inheritance hierarchies. Further possibilities would be to cluster elements belonging to the same package or classes connected by means of aggregation or composition relationships. Figure 6 (right hand side) shows an example for this technique. The visible class C1 is part of a hierarchy located off-screen. All classes which are directly or indirectly sub-classed from class C2 are aggregated into one cluster.

According to geometric clusters, structural cluster elements show the amount of clustered classes by means of a number (in this case six). Again, the number is incremented and decremented when a clustered node becomes visible or invisible respectively. Structural cluster proxies are located at the place where the next connected off-screen node of the cluster is projected. In Figure 6 (case 2) C1 is connected with off-screen class C4 and the cluster proxy appears at the position where C4 is projected by means of orthogonal projection.

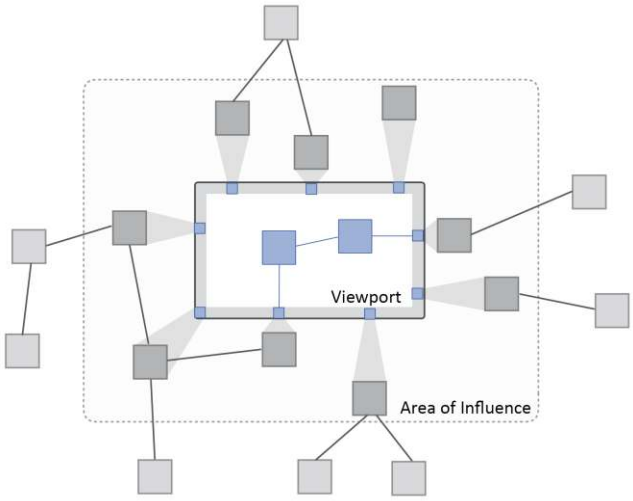

Figure 7. A rectangular virtual *area of influence* is located around the viewport (typically, the screen). All nodes within this area are represented as proxy elements. Nodes outside the area are ignored.

If additional meta-information or semantic information is available this can also be applied for creating clusters. For example, proxies could be clustered if they belong to a certain part of a class hierarchy or to a sub-graph with associated semantic meaning. Another example is the application of meta-information for feature-oriented software development[1]. Thereby, classes attributed to a certain feature could be clustered.

4.2.3 Area of Influence

If diagrams with hundreds or even thousands of nodes are visualized, even the clustering strategies mentioned before may not be applicable. In this case, we suggest that the currently visible part of the diagram within the viewport (e.g., the editor window) is surrounded by a virtual *area of influence* (see Figure 7). The *area of influence* is part of the off-screen area and can have arbitrary shape. Typically rectangular or circular shapes will be used. Only off-screen nodes residing within this *area of influence* are represented as proxies. All other, further away off-screen nodes are ignored and filtered. Further filter techniques are described in Section 5.4. The *area of influence* moves with the viewport during panning and grows and shrinks during zooming (proportional to the applied zoom factor). To always show a predefined maximum number of proxies, the size of the *area of influence* can also vary according to the amount of off-screen nodes located within the area. It gets bigger if the viewport is located in a sparse region of the diagram and becomes smaller if it is located in a dense region. Finally, if the off-screen visualization technique is combined with a traditional overview+detail interface, the *area of influence* can also be indicated within the overview window.

4.3 Design of the Interactive Border Region

For the appearance of a proxy element, there are different design variants conceivable. They depend on the dimension of the border region (see Figure 8). For a one-dimensional (1D) border, proxy elements can be drawn as symbols with different colors, shapes or labels. Their spatial layout and how they are positioned to each other on the two dimensional canvas is not considered by this representation. In particular, approaches such as the onion-graph notation [23] can be applied for clustered inheritance hierarchies in class diagrams. Thereby, proxy elements can be put inside each other to visualize the clustering.

Furthermore, we propose to stack proxies according to their position within the diagram layout. This could be seen as a 1.5D solution, as the spatial position of nodes would be recognizable without a complete 2D layout. Finally, the border region could allow a two dimensional arrangement of proxy elements according to the geometric layout of the associated nodes. This would result in a bifocal view [41] providing a condensed view of the remaining diagram within the border. Furthermore, we propose to use rounded corners for the interactive border. This approach can avoid clustering of proxy elements in the corners of the display if orthogonal projection is applied. Beyond that, for radial projection rounded corners can avoid an abrupt change of direction of proxy elements during panning. These solutions are subject of further investigation.

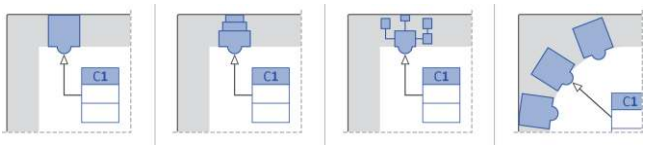

Figure 8. Different dimensions of the border region, from left to right: 1D, 1.5D and 2D. Border region with rounded corners (right).

5. INTERACTION TECHNIQUES

The previous section provided a detailed description of the visualization techniques of our approach. In this section we present how users can interact with the off-screen visualization and how it reacts on user input. Like in common diagram editors we support manual panning (e.g., by mouse dragging) and zooming (e.g., by using the mouse wheel). Thereby, the positions of proxy elements are constantly updated. The update takes place according to the position of the associated off-screen nodes and the applied projection algorithm. Furthermore, when a node crosses the border of the viewport, the respective proxy element is blended smoothly in and out, to make the relation of node and proxy comprehensible. Besides this manual navigation our system also realizes automatic navigation which is described in the following. We start with the appearance of node previews and then the automatic navigation itself is explained. Finally, we present further techniques such as inserting edges via proxies and interactive filtering.

5.1 Preview of Off-Screen Nodes

Hovering with the mouse cursor over a proxy, results in a preview of the associated node. The preview is shown as an overlay within the diagram workspace and is located close to the border region at the side of the respective proxy element. For cluster proxies a list of previews appears consisting of one preview for each clustered node (see Figure 9 left). In our prototype a preview shows the label of the class or interface. Each preview has the same color as the associated proxy element. The previews are blended out smoothly when the mouse cursor is leaving the proxy element. Besides that, if a visible node is selected, the proxy elements

[1] See http://fosd.de/ for further information on feature-oriented software development

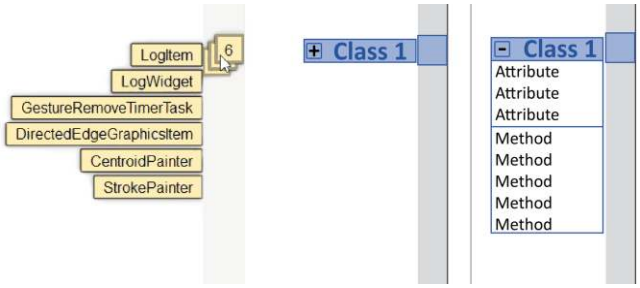

Figure 9. A list of previews is shown if a proxy is hovered (left). Expandable previews (center) can give further information about the content of an off-screen node and allow in situ editing (right).

which are directly connected with the selected node could show their previews automatically. In that way, a user can easily get more information about nodes connected with the currently selected node.

Showing the label of the associated off-screen node is certainly the simplest version of a preview. Previews could also show further details of the content of a node like with semantic zooming [30]. Beyond that, the relationships of nodes could be visualized by previews. Concepts for these approaches are presented in the following subsections.

Content previews. Expanding the previews can give access to the content of the respective node. Figure 9 (center) shows a preview label which is equipped with a "+" button. If it is pressed, the preview is expanded and attributes and methods of the associated class become visible (Figure 9 right). With these previews it is possible to edit off-screen nodes in situ without time consuming zooming and panning. For example, content can be added, deleted or changed by interacting with the expanded preview as with a normal class. If previews become too big due to too much content, several levels of detail could be used.

Topological previews. Besides details concerning the content of an off-screen node, previews can also show relationships between nodes. Thereby, the topology of an off-screen sub-graph is visualized but not its actual layout. Figure 10 shows the topological preview for the example of Figure 6. The list of previews for a cluster is equipped with a button (Figure 10 left). If it is pressed the previews are dynamically rearranged and the complete inheritance hierarchy is visualized within a dedicated area (Figure 10 right). Furthermore, the area can be changed to arbitrary size by dragging its border.

Of course both approaches – *content* and *topological previews* – can be used at the same time. Preview labels shown in a topological preview can be expanded to edit their content.

5.2 Automatic Navigation

In addition to traditional navigation by manual panning and zooming, we offer automatic navigation. This is achieved by clicking a proxy element or a preview which results in an automatic zoom+pan animation to the respective off-screen node. With this technique it is possible to focus a particular node in a targeted and fast way. In particular, users are able to explore the topology of the diagram by hopping from node to node. In UML class diagrams for example, this technique can be applied to navigate within inheritance hierarchies along generalization relationships by clicking proxies which represent connected classes. To make the automatic navigation as smooth and comprehensible as possible, we applied simultaneous panning and zooming according to the algorithm introduced by van Wijk and Nuij [47].

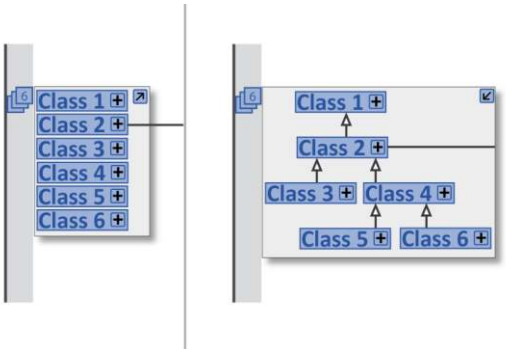

Figure 10. Topological preview of clustered nodes. The preview shows how the nodes are connected.

If a cluster proxy is clicked, the viewport is animated in a way that all clustered nodes are focused. To navigate to a specific node which is aggregated within a cluster proxy, there are two options. Either the respective node is chosen from the list of previews or a double click is performed on the cluster proxy. By means of the double click the cluster proxy is expanded in an animated way, showing all clustered elements as single proxies. For geometric clusters the expanded proxies are distributed evenly in the neighborhood of the cluster. For structural clusters all associated nodes are projected by means of geometric projection resulting in proxy elements at the respective location.

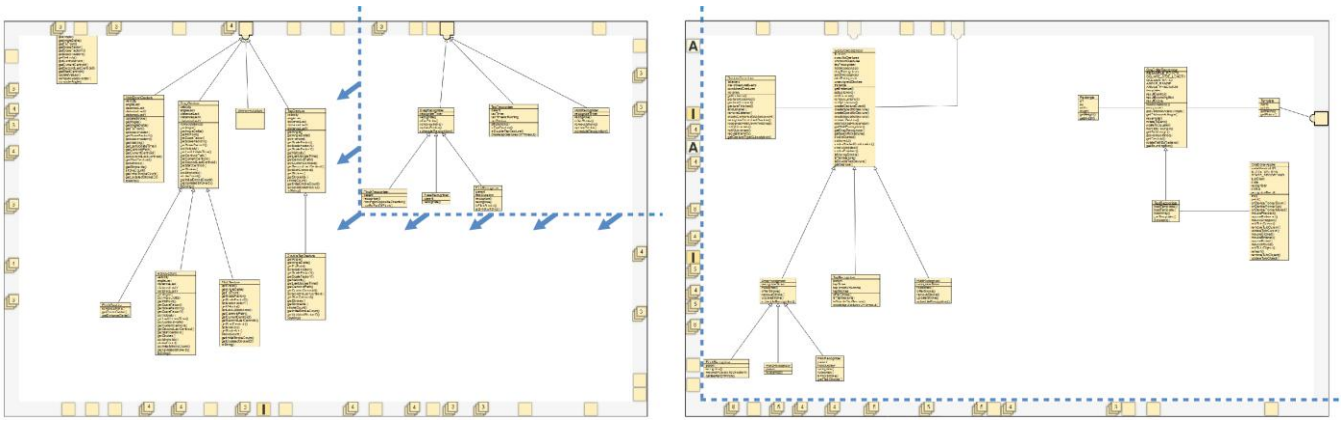

Figure 11. Two screenshots of our prototype: A particular part of the class diagram is focused (left). The position of the proxies is dynamically updated during panning and zooming. For example, panning the view to the left and down (indicated by the blue dashed line and the arrows, added to the Figure for illustration), results in the screenshot at the right hand side.

5.3 Inserting Edges

Besides providing a context visualization and quick navigation to clipped nodes, our technique also supports basic editing. Edges can be created between visible nodes and off-screen nodes. Thereby, an edge is dragged from the respective node to a proxy element of the border region. As a result, it is connected automatically with the associated off-screen node. Thereby, the inserted edge is connected with an already existing edge port or the edge port appears when the edge is dragged on top of the proxy element. In that way, labels and other properties such as multiplicities can be edited in place without further panning and zooming. If the edge is dragged on top of a cluster proxy, previews for all containing nodes are shown. An edge can be created by dragging it to the particular preview. However, other nodes can be located in the way of the inserted edge. Therefore, an automatic edge routing which avoids the crossing of nodes such as described by Wybrow et al. [50] should be applied.

5.4 Interactive Filtering

In addition to automatic clustering we propose interactive filtering of proxy elements to prevent clutter and to make our technique scalable to large diagrams. Filter criteria can be adjusted interactively by means of user interface widgets. As a result, proxies not meeting the applied criteria are blended out. There is a variety of filter criteria conceivable. For example, proxies can be filtered according to their type (e.g., only proxies representing abstract classes are shown), according to their topological distance from the focused node (e.g., only proxies of directly connected classes are shown) or according to particular metrics (e.g., only proxies of god classes [33] with a huge amount of attributes and methods are shown).

6. THE PROTOTYPE

We implemented the off-screen visualization approach as a prototype for navigating and editing UML class diagrams. Figure 11 shows two screenshots of the prototype as it was also used for the user studies presented in Section 7 and Section 8. In the following subsection we describe the implemented features and the basic algorithmic approach for the off-screen visualization.

6.1 Implemented Features

The application is written in Java whereby the graphical user interface is based on Qt Jambi. For keeping a consistent graph structure the open source toolkit jGraph [26] is used. Furthermore, the prototype is based on the Eclipse UML model [9] and diagrams can be imported by means of XMI. Besides creating layouts in a manual way, it is also possible to apply automatic layout algorithms offered by jGraph and Zest [52]. The layout of a diagram is stored in a separate file, using our own XML format.

The class diagram is shown in the center region. Proxy elements for off-screen nodes are placed within the interactive border region (depicted with light gray background in Figure 11). Users are able to pan by dragging with the mouse (holding the left mouse button pressed) and to zoom with the mouse wheel. Proxy elements are dynamically updated during interaction.

Our prototype is capable of visualizing class diagrams consisting of classes, abstract classes and interfaces. Concerning relationships, associations (directed and undirected), generalizations and realizations can be shown. However, edge labels and multiplicities are not yet visualized. All nodes are represented by respective proxy elements. Their appearance is shown in Figure 5. We realized both ways of geometric projection (orthogonal and radial) and *along edge projection* as explained in Section 4.1.2. For geometric projection, the change of edge routing is performed by inserting a *proxy edge segment* from the last visible inflection point to the respective proxy. Proxy elements are clustered when two or more proxies are created at the same position (see Figure 5 for cluster icon). Furthermore, we implemented a simple algorithm for avoiding clusters (see Section 4.2.1). If there is enough space available proxies are placed side by side until a certain distance threshold is reached. Besides that, we implemented structural clustering for inheritance hierarchies. If parts of a hierarchy are located off-screen they are aggregated in a cluster. When proxies are hovered with the mouse cursor, labels of the associated classes or interfaces are shown as previews. The previews are blended out smoothly with a one second delay after the mouse has left the proxy or disappear immediately if the background is clicked.

We also realized temporal geometric projection for *along edge projection* (as described in Section 4.1.2).

6.2 Implementation Details

In our prototype the border for the off-screen visualization is a separate user interface component which encapsulates the algorithms for visualizing the proxy elements.

As an initial step, all nodes are determined which are currently rendered completely (with their whole bounding rectangle) in the viewport. They are marked as on-screen. Furthermore, nodes intersecting the viewport and all nodes currently not rendered are labeled as off-screen. Then proxy elements are created for each off-screen node. The position of the proxy is determined by projecting the center of the respective node onto the border of the viewport. This is achieved by intersecting the border with the line through the nodes' center to the center of the viewport (for radial projection) or perpendicular to the viewport (for orthogonal projection).

The basic algorithm is divided into three steps: determine on-screen and off-screen nodes, updating the positions of the proxies and creating and updating the clusters.

Determine on-screen and off-screen nodes. For each viewport update (panning and zooming) at step *n* the currently visible nodes are determined. This snapshot is compared to the visible nodes of step *n-1*. Nodes which are completely visible at step *n* but were not visible at step *n-1* moved from the off-screen area to the on-screen area. Their proxies are deleted (possibly removed from clusters) and they are marked as on-screen. In contrast to that, nodes which are not visible anymore or just partly visible are marked as off-screen.

Updating the positions of the proxies. As a next step, the positions of the proxies are updated by performing the respective projection. In our algorithm proxies with attached edges (associated to nodes connected with on-screen nodes) are updated first. They are preferred, as lagging is particular noticeable for them when they fall behind their faster moving edges. During this step, *along edge projection* is performed, whereby the proxy is positioned at the location where the edge intersects the viewport.

After that, all unconnected proxies are updated. If geometric clustering is enabled and orthogonal projection is applied, it is not necessary to move every single proxy. If proxies are clustered, it is sufficient to update the position of the whole cluster (as proxies once clustered, never leave the cluster and move along with it).

For radial projection, proxies associated to nodes located closer to the viewport move faster than proxies for nodes positioned further away. Therefore, proxies which were overlapping at step *n-1* do not necessarily have to overlap at step *n*. Therefore, geometrical clusters can change and the position of each proxy has to be updated separately. This makes radial projection more computationally expensive.

Creating and updating the clusters. Proxies for nodes which moved off-screen are added to structural clusters if certain conditions are met (e.g., the node is part of an inheritance hierarchy). If the proxy is not part of a structural cluster and if it is not connected with an on-screen node, the system checks if it overlaps with other proxies. If an overlap exists, the proxies are aggregated in a geometrical cluster.

The off-screen visualization component runs in its own thread to allow smooth interaction. With the approach described above, we are able to navigate diagrams with up to 400 nodes without performance issues.

7. Pilot Study

We conducted two studies. First, we ran a pilot study with our early prototype. Our goal was to collect feedback at an early stage of development, to come to decisions for further design iterations. After that, we conducted a controlled user evaluation with the prototype modified according to the results of the pilot study. It is presented in Section 8.

In particular, with the pilot study we wanted to clarify the following questions: Are people able to understand the visualization technique spontaneously? Which kind of geometric projection is preferred – orthogonal or radial projection? Are the proxies properly designed and distinguishable from each other? Is *along edge projection* comprehensible? We applied a think-aloud approach in combination with user observations and a questionnaire.

7.1 Design of the Study

Apparatus. The evaluation was conducted with the prototype mentioned in Section 6. It ran on a PC with 2.5 GHz and 3 GB RAM under Windows XP. The display had a resolution of 1680x1050 pixels and a screen size of 20'', because we considered this as common for average workplaces.

Participants. Eight participants (6 male, 2 female, age from 24 to 35) took part in the evaluation (6 employees of the computer science department, 2 graduate students). They all have a solid background in computer science, visualization or HCI. They were not modeling experts, but knew UML class diagram notation and used respective editors from time to time.

Tasks and procedure. Before the evaluation procedure started, the basic approach of the off-screen visualization was explained. This was done by means of the prototype and a small example diagram consisting of ten nodes and six edges. We explained the zoom+pan navigation, the meaning and appearance of proxy elements and the interaction with proxies (hovering and automatic navigation). However, we did not explain further details such as projection or clustering strategies. Orthogonal projection for unconnected nodes and *along edge projection* for connected nodes was used. The whole introduction took about 5-10 minutes.

The evaluation procedure was structured in two parts. Part one consisted of a guided navigation within a smaller class diagram. This means, we asked the participants to navigate to particular nodes by clicking on proxies and guided them on a way through the diagram. During the procedure we asked them about their opinions concerning certain design issues and logged their comments and behavior.

Before they started to use the prototype, a printout of the UML class diagram was handed to the participants. The structure of the diagram was explained to them, and they were asked to memorize the spatial layout of the diagram for 1-2 minutes. To make its content easily understandable, the diagram modeled the

structure of a theater. For example, there were classes named *actor* and *stage play*. An *actor* plays a role within a *stage play* which was expressed by an association. Furthermore, a stage play is a special kind of *event* – expressed by a generalization. The diagram consisted of 31 classes (3 of them abstract) and 35 relationships (18 associations and 17 generalizations). The diagram was layouted manually according to aesthetic rules [10]. For instance, general classes were always located above their subclasses, crossing of edges was avoided and classes belonging together on a semantic level were also located close together in the layout.

Every participant started the guided navigation at the same position and followed the same navigation path given by our instructions. In particular, we asked the participants to perform several smaller tasks. We asked them to estimate the direction of a class located off-screen, to indicate an off-screen class on the printout without using the previews and to navigate to a certain class and tell its directly connected neighbors. Furthermore, we asked them to count abstract classes to see if proxies are distinguishable from each other. At a certain point of the navigation a temporal projection (see Section 4.1.2) occurred, as the respective class was connected by means of a bent edge. We asked the participants if they could explain this behavior spontaneously and discussed this technique. At the end of part one, participants were asked to explicitly compare geometric projection and *along edge projection*. For that, they were asked to navigate freely in both modes. To clearly demonstrate the creation of several proxies for one class in *along edge projection* mode, a class with eight edges was used. For each edge one proxy was created.

In part two, the participants were asked to freely explore an unknown UML class diagram consisting of 72 classes, 8 interfaces and 89 relationships (30 associations, 45 generalizations and 14 realizations). The exploration had a duration of approximately five minutes. Subsequently, we demonstrated the *avoid cluster algorithm* and asked the participants if it is comprehensible to them.

During both parts, we took notes about our observations, and the participants' comments and suggestions. Beyond that, at the end we handed a questionnaire to them with four questions. For example, they were asked to rate the discriminability of proxy elements and the comprehension of automatic zoom+pan on five point Likert scales (from 1 = "completely disagree" to 5 = "completely agree").

7.2 Results of the Pilot Study

Navigation. All participants quickly understood the basic approach of the off-screen visualization technique. However, for the first navigation task most of them spontaneously applied traditional zooming and panning. After an additional hint that navigation is also possible by clicking on respective proxy elements, participants mainly applied this approach. Especially, two participants emphasized that they liked the idea of "navigating the diagram step-by-step" by clicking proxies and jumping from node to node.

Participants commented that zoom+pan animation was too quick and should zoom out more during panning to give a decent overview. Hence, the comprehensibility of the animation was rated with a rather low mean value (M = 1.6, see Figure 12). However, the animation parameters can be easily adjusted. Furthermore, two participants remarked that they would not need a smooth animation at all, as their only attempt is to quickly navigate to the associated node.

Projection. Most of the participants (6 of 8) expected radial projection and were not able to identify off-screen nodes correctly without using the preview function. The question if off-screen nodes were located at the expected position was rated with a mean value of 3.0 (see Figure 12). Furthermore, after explaining the principle of *along edge projection* was comprehensible to the participants. Most of them liked the idea of maintaining the routing of edges. However, many participants mentioned that the occurrence of several proxies for the same node is confusing and suggested a clearer indication which proxies are associated to the same node. Similar results were collected for the temporal projection. It was understood by the participants after explanation, but they suggested a clearer indication of temporal proxies.

Appearance of Proxies. Proxy elements representing classes directly connected with visible nodes were clearly distinguishable from other proxy elements. The discriminability of proxy elements was rated with a mean value of M = 2.3 (see Figure 12). As mentioned in Section 4.2, the color of the proxies matched with the color of the respective node. Many participants suggested using different colors which are more distinguishable from each other. However, all participants were able to identify the different types of proxy elements when they were asked to count proxies representing abstract classes and interfaces. Furthermore, five participants suggested adding more information to the proxies, such as the amount of methods or attributes of a class.

Further observations and comments. One participant suggested a history function as suggested in [38], to navigate back to previously visited nodes. This can be beneficial if a proxy was clicked by accident or if navigating back is necessary during the editing process. Furthermore, three participants asked for a distance indication. As previously mentioned, we assumed this as less important for the domain of node-link diagrams. For which tasks distance indication is beneficial and how it can be achieved in combination with our approach is subject for further investigation. Moreover, six participants asked for an overview, and we observed that all participants used the printout of the diagram for orientation. The orientation within the diagram was rated with a mean value of M = 3.1 (see Figure 12). In fact, an overview was already implemented for the editor but we turned it explicitly off for the evaluation. In which way an overview supports our approach is studied in the evaluation presented in Section 8.

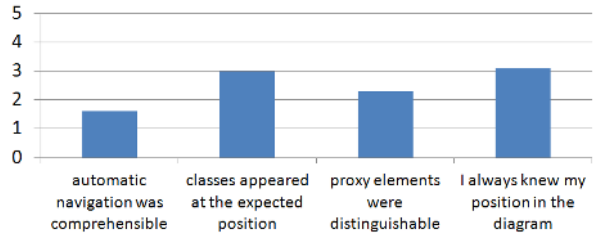

Figure 12. Results of the pilot study questionnaire

7.3 Adoptions resulting from the Pilot Study

Based on the observations and comments we collected during the pilot study, we changed our prototype in several ways.

Along Edge Projection. Due to *along edge projection* some nodes (with several edges) were represented by several proxies. As we have found in the pilot study, some participants were confused especially, when a node was represented by many proxies. To mitigate this problem, we slightly adopted the approach used in the pilot evaluation. Both projection techniques – geometric and *along edge projection* – are now applied simultaneously. An example is illustrated in Figure 13. Geometric projection is used for nodes connected with straight edges such as the generalizations from C1 to C2 and C3. In this case the change of edge routing is rather easy to comprehend, and it is ensured that there is just one proxy for the node (instead of two). *Along edge projection* is applied only for nodes connected with bended edges to prevent confusing changes of edge routing. In Figure 13 this is the case for the association between class C1 and C4. Furthermore, proxies created by *along edge projection* are drawn in a semi-transparent style to distinguish them from geometrical projected proxies.

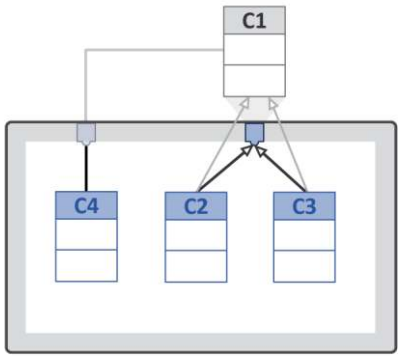

Figure 13 Combination of along edge projection (for bended edges) and geometrical projection (for straight edges). Class C1 is represented by two proxies instead of three.

Temporal Projection. To indicate *temporal projection* more clearly we decided to visualize the routing of edges temporally within the border region. An example for this is illustrated in Figure 14 left. If one of the proxies created by *along edge projection* (1` or 1``) is hovered with the mouse cursor, a temporal projected proxy appears (1*) and the routing of edges is indicated by *proxy edges* leading to the temporal projected proxy. If the *proxy edge* crosses other proxies they are grayed out to prevent clutter within the border region (see Figure 14 right). In that way it is clearly visualized how the visible nodes are connected with off-screen nodes and which nodes are represented by several proxies.

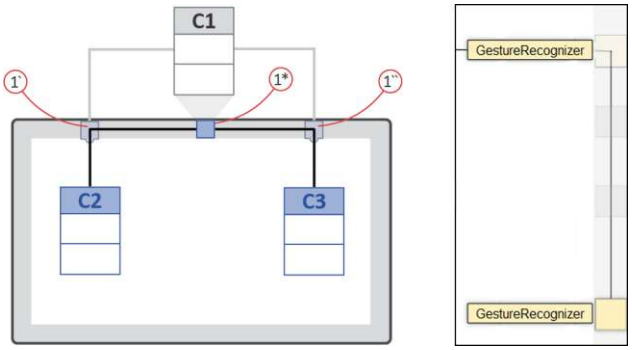

Figure 14. Left: If one of the proxies is hovered, proxy edges are shown within the border region to clearly indicate that one node is represented by several proxies. Right: Screenshot from the prototype. If a proxy edge crosses other proxies they are grayed out to make the routing clearly visible.

Clustering. We added animation to the proxies to make the creation of geometric clustering (see 4.2.1) more comprehensible. For that, we decoupled the movement of proxies during panning and the creation of clusters from each other. During the panning process the position of proxies is dynamically updated and proxies can overlap or even occlude each other when they move within the border region (see Figure 15, left and center). This happens especially when radial projection is applied as proxies representing nodes closer to the viewport move faster than proxies of nodes located farer from the viewport. A drop shadow was added to proxies to make the overlapping clearer. When the user stops panning, overlapping proxies are animated towards each other. When they completely overlap, they are deleted and a cluster icon is blended in smoothly (see Figure 15, right). In that way, updating the position of proxies during panning and the creation of clusters are decoupled from each other.

Figure 15. Screenshot of the prototype: During panning proxy elements (left) can overlap (center). If the user stops panning the overlapping proxies are animated towards each other and a cluster proxy appears (right).

History Function. We also added a history function for the automatic navigation by clicking on proxies. In contrast to traditional undo functions (e.g., Ctrl+Z), it can be applied to quickly navigate back to previous views. Other activities such creating or editing diagram elements are not affected. The history function can be invoked by holding a keyboard shortcut (e.g., the shift key in our prototype). As a result the proxies for the last five visited nodes are highlighted; all other proxies are grayed out. In that way users are able to quickly see the recently visited nodes and to jump back to them directly by clicking. After they have navigated back to a particular node, the history function can be invoked again (and the last five visited nodes are highlighted again). This technique allows going back in navigation history with a maximum step size of five steps. We chose to initially visualize the last five nodes to reduce the

cognitive burden of the user. However, we suggest setting the amount of highlighted proxies (and thereby the maximum step size) dynamically by mouse dragging. For example, by activating the history mode, pressing a mouse button and dragging the mouse horizontally users could adjust the number of highlighted proxies. Moving the cursor from left to right blends in more proxies of the recently visited nodes according to their position in the history. This can be done until all proxies are highlighted or the end of the complete history is reached. Moving the cursor in the opposite direction reduces the amount of highlighted proxies.

8. User Evaluation

With the improved prototype we conducted a controlled experiment to evaluate the performance of our approach more deeply. Our goal was to investigate to what extent the off-screen visualization improves diagram navigation concerning speed and user satisfaction. Beyond that, we wanted to find out if users are able to stay oriented within an unknown diagram while navigating by clicking on proxies. Therefore, we ran a study with three conditions (see Figure 16). We compared a zoom+pan interface with overview+detail as applied in state-of-the-art diagram editors (condition ***OD***, Figure 16, top) and the off-screen visualization technique without overview (condition ***OS***, Figure 16, center). We expected that OS users will be faster than OD users due to automatic navigation by clicking proxies. Furthermore, we expected that the automatic zoom+pan animation will support the participants' orientation as good as manual navigation. In addition to that, we ran the evaluation with a user interface realizing the off-screen visualization technique combined with an overview window (condition ***OS+OD***, Figure 16, bottom). We expected that the presence of an overview window will improve the orientation within an unknown diagram as participants have an additional view to easily spot their location. However, we did not expect that an additional overview will lead to a better performance concerning navigation. This was based on the expectation that users will stick to navigation by proxy elements even if an overview is present. Furthermore, we investigated how precisely users can navigate to a given off-screen node by using automatic navigation.

To summarize, our hypotheses were as follows:

H1 Using the off-screen visualization (conditions **OS** and **OS+OD**) is faster than using the overview+detail interface (condition **OD**).

H2 For the off-screen interface, navigation task completion time is not influenced by the presence of an additional overview window (see conditions **OS** and **OS+OD**).

H3 With regard to orientation tasks, there is no influence of the **OS**-interface (with automatic zoom+pan animation) compared to the OD-interface (with manual zoom+pan and overview).

H4 The off-screen visualization combined with an additional overview view window (condition **OS+OD**) improves the orientation in comparison to the **OS** and the **OD** interfaces.

8.1 Design of the Study

For the study we applied a between subjects design. There were three groups of participants – one for each interface.

Participants. 27 voluntary participants took part in the study (aged 23-39 years, 7 female, one left handed). Most of them are faculty members of the computer science department and six of them are advanced students of higher semesters. None of them took part in the pilot study and none of them knew the UML diagrams and their content. They are no everyday modelers but only two of them had no knowledge of UML class diagrams.

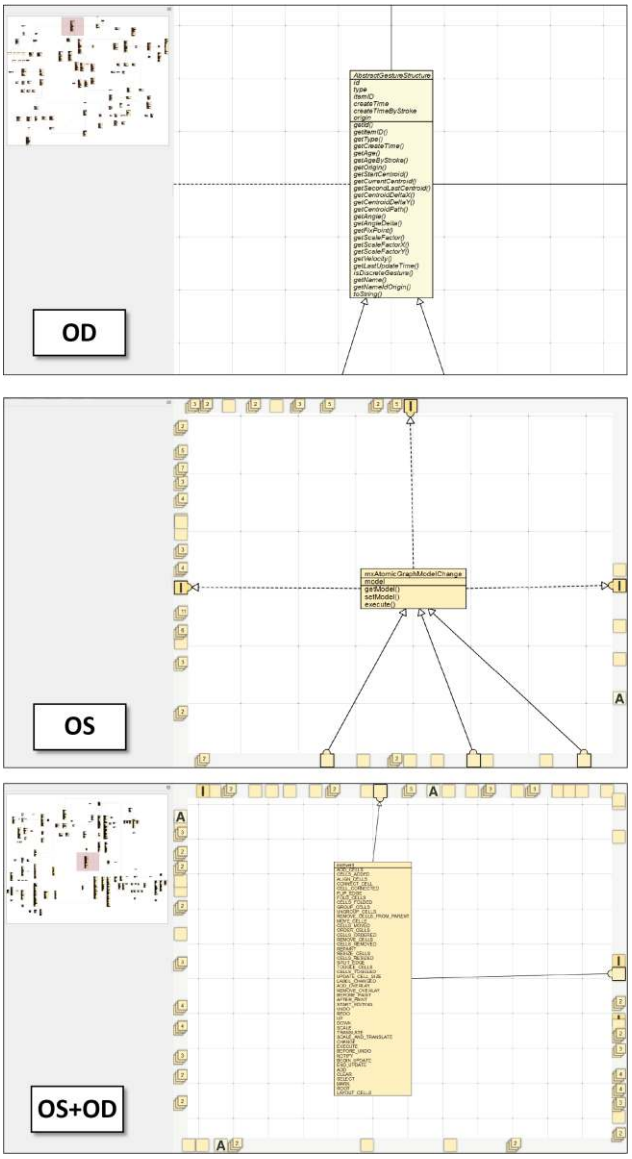

Figure 16. Screenshots of the interfaces for the three conditions: *OD* (top), *OS* (center) and *OS+OD* (bottom). A red viewfinder rectangle indicates the position of the viewport within the overview.

Two participants give lectures on software modeling and five participants stated that they are regular users of other notations (such as activity or dataflow diagrams) and respective editors. UML and diagram editor expertise was determined before the study by an online questionnaire with two five-point Likert scales. We considered these results to equally assign participants according to their knowledge to the three conditions.

Apparatus and Interfaces. The study was conducted at a PC running with 3 GHz, 8 GB RAM and Windows 7 (64 bit). The display had a resolution of 1900x1200 pixels and a screen size of 24''. For both off-screen conditions our prototype application was used, including the adoptions mentioned in Section 7.3 (just the history function was disabled). Based on the results of the pilot study the proxy elements were created by radial projection and the zoom+pan animation was adjusted. It was made slightly slower and zoomed out a bit more to show more of the diagram during animation. The three interfaces are shown in Figure 16. The width of the border region was 45 pixels. For the *OD* and *OS+OD* interfaces the overview window occupied about 15% of the whole display space (similar to the conditions of Nekrasovski et al. [27]). The representation of the diagram within the overview window was too small to read labels, but edges were still visible. Zoom+pan interaction was activated in all conditions. Panning was possible by dragging with the mouse on the background, and zooming could be achieved by scrolling the mouse wheel. Participants could also interact with the overview by dragging the viewfinder rectangle or by clicking at the respective position. Beyond that, the position and size of the viewfinder was smoothly animated during zoom+pan animation in the *OS+OD* interface.

Datasets. Two UML class diagrams of similar size served as datasets for the study. The first one (D1, see Figure 19 in the appendix) was a class diagram of a multi-touch gesture recognizer developed in-house (90 nodes, 84 edges). The second diagram (D2, see Figure 20 in the appendix) showed parts of the graph visualization toolkit *mxGraph* [26] (99 nodes, 103 edges). We used the same representation of class diagrams as in the pilot study: the diagrams consisted of classes, abstract classes (5 in D1 and 8 in D2, with labels in italics) and interfaces (5 in D1 and 10 in D2). Relationships were limited to associations, generalizations and realizations. Furthermore, there were no labels for relationships. Therefore, participants needed no deep expert knowledge in UML. We manually layouted the diagrams according to aesthetic rules [10] as in the pilot study.

Tasks. The tasks were divided into three blocks. The first block consisted of two simple and two more complex *comprehension tasks*. Thereby, we asked the participants to analyze relationships within a UML class diagram. These tasks were followed by a second block of *orientation tasks*. Participants were asked to go back to already visited nodes to find out if they were able to stay oriented during navigation. Both task blocks were conducted with the two class diagrams (D1 and D2) mentioned before. With these tasks we simulated the situation that a user wants to edit parts of an unknown diagram. For that, he/she has to understand several relationships starting from a particular node of interest (task block 1). After that, the user has to navigate back to the location he/she started from to edit the content (task block 2).

The third block consisted of three *locate tasks* which were performed only by participants doing the off-screen conditions (*OS* and *OS+O*). To automatically navigate to a particular off-screen node, participants had to choose the respective proxy. In contrast to the tasks of block one and two, the *locate tasks* assumed that the participants had to orient themselves within a known diagram. As the applied diagram was unknown to them, we simulated this situation by telling them the name and direction of the requested off-screen node – all information users would have if they would be familiar with the diagram. The tasks are described in more detail in the following sub-sections.

Comprehension tasks were divided in two simple and two complex tasks. For the *simple comprehension tasks* **(SCT)** we asked which classes are implementing a focused interface (or vice versa: which interfaces implement the focused class). For that, the participants were asked to navigate to this node. Users then had to name the classes or interfaces connected by realizations and to navigate to a particular one of them. Both of these simple tasks differed in the amount and the distance of adjacent classes or interfaces. For the *complex comprehension tasks* **(CCT)** we asked the participants to name all super classes of a focused class. So users had to navigate from the lowest level of the inheritance hierarchy to the top. After that, they were asked which interface implements the topmost class. Again, there were two of these complex tasks. They differed in the amount of super classes to find (four and five, respectively) and the amount of associations leading from the classes. For example, in one case the root class was connected with several associations so that participants had to check if they already reached the top of the hierarchy.

Task Block 1	
1.1 SCT1 and SCT2	Two sequential simple comprehension tasks: Which classes implement the focused interface? (Or: Which interfaces realizes the focused class?) Name all of them and navigate to class/interface X.
1.2 CCT1 and CCT2	Two sequential complex comprehension tasks: Name all super classes of the focused one. Which interface implements the topmost class? Name it and navigate to this interface.
Task Block 2	
OT1, OT2 and OT3	Three sequential orientation tasks: Navigate back to class/interface X as fast as possible.
Task Block 3	
LT1, LT2 and LT3	Three sequential locate tasks: Choose the proxy element for the indicated class/interface.

Table 1: Summary of the tasks

For the *orientation tasks* **(OT)** we asked the participants to navigate to two particular classes and one interface as fast as possible. All these nodes had been visited before and were classes or interfaces where the *comprehension tasks* started. Before they started with the respective task, we asked the participants if they could remember the direction of the target node. We explained this task to them beforehand. However, we did not explicitly encourage them to memorize the navigation path while doing the tasks of the first block. In that way, we could see if they were able to stay oriented spontaneously.

For the three *locate task*s **(LT)** (*OS* and *OS+OD* only) the participants were asked to choose the proxies for two off-screen classes and one interface. We told them the names of the respective target nodes and manually indicated their positions within the overview window. Participants had to estimate the direction and to find the proper proxy element by hovering it with the mouse cursor. The two off-screen classes were located at the upper right and lower right, respectively. The off-screen interface was located towards the left of the current viewport.

Procedure. At the beginning of the study we explained the simplified UML class diagram notation to the participants. After that, we demonstrated the interaction techniques to them. For each condition we explained zoom+pan and the functionality of the overview if present. For *OS* and *OS+OD* conditions we explained the off-screen visualization in detail. This comprised types of proxies, docking of edges, zoom+pan animation, creation of clusters and along edge projection. After that, participants trained the *comprehension and orientation tasks* with a small class diagram consisting of 23 nodes and 25 edges. The duration of training was about five minutes.

Comprehension and orientation tasks were performed within a fixed order for each of the three conditions – first with diagram D1 and then with diagram D2. After that, participants of the *OS* and *OS+OD* conditions performed the three *locate tasks*. For the OS condition the overview was activated to manually indicate the target classes and interface. For off-screen conditions (*OS* and *OS+OD*) a session had a mean duration of 30 minutes. For the *OD* condition the duration was shorter (22 min) due to shorter explanation and training phases.

Measurements. For block one and two we measured the completion time for each task. For block three we counted the attempts participants needed to find the proper proxy element. Furthermore, we noted comments and observations during the study. In the end, participants were asked to rate the difficulty of the tasks and the usability of the interface on five-point Likert scales.

8.2 Results

For the *comprehension* and *orientation tasks* we ran one-way independent ANOVAs. The Bonferroni adjustment was used for post-hoc comparisons. For off-screen interfaces (conditions *OS* and *OS+OD*) some values were discarded as participants switched to manual panning and zooming instead of using automatic navigation by clicking proxies. This happened in only three cases. Furthermore, in two cases tasks were not done correctly (participants clicked proxies instead of just reading the names).

For non-homogeneous variances we performed Kruskal-Wallis tests (with Man-Whitney post-hoc tests). Results of the questionnaire were mapped to a scale ranging from 0 (completely disagree) to 4 (completely agree).

Comprehension Tasks. For the overall completion time of the *comprehension tasks* (see Table 1, Task Block 1) we found significant effects for both diagrams (D1: $F(2,24)=10.869$, $p<.001$, D2: $F(2,24)=22.3$, $p<.001$). For D1 and D2 users with the off-screen interfaces were significantly faster than participants using the *OD* interface. However, there was no significant effect between the *OS* and *OS+OD* conditions for both diagrams. These results confirmed our hypotheses **H1** and Fehler! Verweisquelle konnte nicht gefunden werden.. Furthermore, after comparing the completion times of D1 and D2, we did not find any learning effects between both diagrams. All tasks were solved correctly. The only exception was one participant who used the OS+OD interface. He announced the wrong class for CCT2 of D2. Beyond that, one OS user navigated to the wrong class for SCT2, but he recognized the mistake and corrected it.

The task completion times for the individual *comprehension tasks* (condition x task) are shown in Figure 17 (top). A closer look at the individual completion times revealed that for diagram D1 there were significant effects for the *simple comprehension task* (see 1.1 in Table 1) SCT1 ($F(2,22)=10.397$, $p<.001$). This means, for both off-screen conditions users were significantly faster (*OS*: $p<.001$, *OS+OD*: $p<.003$) than *OD* user. Beyond that, there was a significant effect for the *complex comprehension task* CCT2 (see 1.2 in Table 1). However, the variances were non-homogenous here: $H(2)=10.727$, $p<.005$. Users of the *OS+OD* ($U=10.0$) and the *OS* ($U = 5.0$) interface were significantly faster than *OD* users.

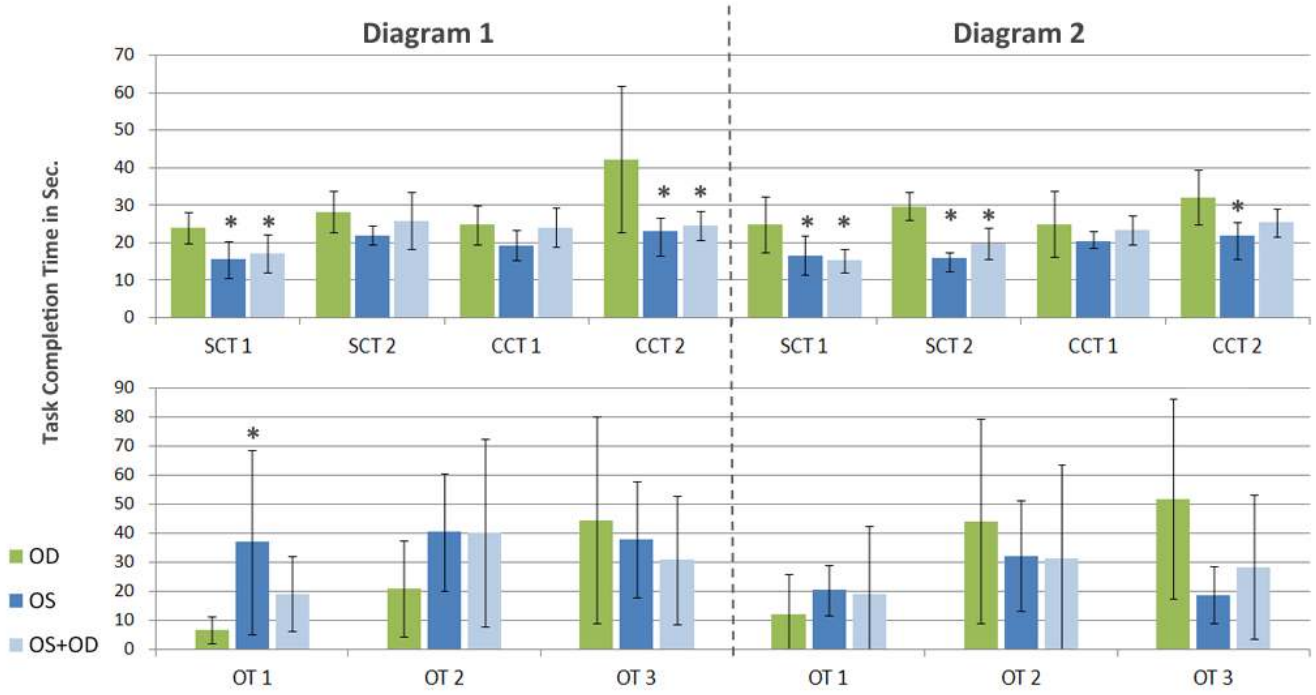

Figure 17. Task completion times for Diagram D1 (left) and Diagram D2 (right). Top: Task completion times for Task Block 1 (SCT = Simple Comprehension Task, CCT = Complex Comprehension Task), Bottom: Task completion times for Task Block 2 (OT = Orientation Task), asterisks mark significant effects compared to the OD interface.

For diagram D2 we found three significant effects for both *simple comprehension tasks* (see 1.1 in Table 1). For SCT1 ($F(2,24)=7.784$, $p<.002$) users of both off-screen conditions performed faster (*OS*: $p<.01$, *OS+OD*: $p<.004$) than users of the *OD*-interface. Similar results were found for SCT2 ($F(2,22)=39.034$, $p<.001$). Again, both off-screen conditions were significantly faster (*OS*: $p<.001$, *OS+OD*: $p<.001$). Finally, there was a significant effect for the *complex comprehension task* (see 1.2 in Table 1) CCT2 ($F(2,23)=7.772$, $p<.003$). For this task, participants using the *OS* interface were significantly faster than the *OD* interface (*OS*: $p<.002$).

Concerning the results of the questionnaire, *comprehension tasks* were rated as relatively easy to solve for all three conditions: M = 3.8 (*OD*), M = 3.9 (*OS*), M = 4.0 (*OS+OD*). There were no noteworthy differences for the ratings of simple and complex tasks.

Orientation Tasks. Figure 17 (bottom) shows the individual completion times for the *orientation tasks*. We found a significant difference between the completion times of the *OD* and *OS* interface for OT1 in diagram D1 (($H(2)=9.348$, $p<.009$). In this case, the OD interface performed significantly faster ($U=5.0$). All other differences were not significant. Furthermore, several participants did not complete all orientation tasks. In most of the cases they gave up on task OT3. For diagram D1 it was canceled three times for the *OD* interface. Beyond that, it was canceled once for the *OS* interface and two times for *OS+OD* interface. For diagram D2 the task OT3 was canceled two times for the *OD* interface and two times for the *OS+OD* interface. OT2 was canceled only once for D2 and the *OS+OD* interface.

Completion times for the individual orientation tasks show that for the *OD* interface participants became continuously slower for both diagrams. This is not surprising, as it was difficult for them to remember the locations of already visited diagram elements over time. For both off-screen conditions the change of completion times is less extreme. However, their mean value is quite high with about 30 seconds. In contrast to the *comprehension tasks*, the *orientation tasks* were rated as more difficult: M = 1.0 (*OD*), M = 1.8 (*OS*) and M = 1.7 (*OS+OD*). Many participants gave comments such as "I cannot remember where I have been" and searched at locations they never visited before. Altogether, these results falsify our hypotheses **H3** and **H4**.

One exception is the *OS+OD* completion time of OT1 for diagram D1. In this case, five participants purposefully used the overview to jump to the target node directly. This lowered the task completion time. A rather low value can also be found for OT 3 (D2) of the *OS* condition. For this task, participants should navigate back to an interface. Three of them searched explicitly for interface proxies and ignored other types of proxies which resulted in lower task completion times. For the *OS+OD* interface only one participant used this approach.

Locate Task. During the locate tasks of task block 3 we asked *OS+OD* and *OS* users to choose the proper proxy elements for indicated classes and interfaces. Figure 18 shows the mean values of attempts for solving these tasks. A dependent ANOVA revealed no significant effects. The overall mean value was 2.15 attempts.

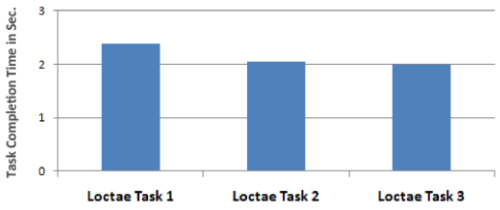

Figure 18. Number of attempts for the *locate tasks.*

Questionnaire and comments. Concerning user satisfaction, the off-screen conditions were rated rather well. Participants stated that the technique was easy to learn (M = 3.4) and easy to use (M = 3.8). Furthermore, they had fun using the system (M = 3.6) and could imagine to apply it regularly (M = 3.6). Participants stated that our approach is a "good technique to handle the complexity of large diagrams" and that "it is better than zooming in and out manually". Beyond that, the different types of proxy elements were clear to them (M = 3.1), the automatic zoom+pan animation was conceivable (M = 3.6) and the creation of clusters was comprehensible (M = 3.4). However, the current position within the diagram was not always clear to the participants using the *OS* interface (M = 1.6, *OS*+*OD*: M = 3.3) and they wished to have an overview (M = 3.9).

8.3 Discussion

Comprehension tasks. The results of the study showed that for exploring relationships within an unknown diagram (*comprehension tasks*) our off-screen technique is at least as fast as state-of-the-art interfaces (zoom+pan in combination with an overview). In more difficult situations – where nodes have several edges of different type or relevant edges are rather long and bent several times – our technique outperforms traditional zoom+pan interfaces significantly. For example, this was the reason for the significant effects of the tasks CCT2 for both diagrams. For both tasks, users of the off-screen conditions were faster. This confirmed our hypothesis **H1**. Furthermore, the off-screen visualization was easily understood by the participants and the majority applied it successfully after a short period of training. For the first task – SCT1 of diagram D1 – off-screen users were already significantly faster. An overview does not affect performance in this kind of tasks which confirmed our hypothesis Fehler! Verweisquelle konnte nicht gefunden werden.. Participants extensively applied our technique and did not pay attention to the overview.

Orientation Tasks. We rejected hypothesis **H3**. The completion times of the orientation tasks and the respective results of the questionnaire revealed that it is quite difficult with our off-screen technique to stay oriented within the given diagrams. Initially, during the *orientation tasks* (OT1-OT3) with the *OD* interface participants were slightly faster in going back to an already visited location. We see the reason for this in the fact that when using a traditional zoom+pan interface, the navigation is performed by the users *themselves*. Therefore, at least the last navigation steps are easier to memorize. In contrast to that, our approach applies *automatic navigation* by animation. This means, users "are navigated" by the system which makes it difficult to recap former navigation steps. We see this as the reason for the rather high drop-out rate for OT3 in both diagrams. However, results of the questionnaire showed that the zoom+pan animation is conceivable and participants stated that they found it beneficial. For example, one participant mentioned that he "would not trust the interface to navigate to the proper node" without animation.

To solve the *orientation tasks*, participants searched for the proper proxy which can take quite some time depending on the size of the diagram. This behavior resulted in the rather high completion times. Altogether, only four participants utilized the different types of proxies and explicitly searched for an interface proxy when they were asked to navigate to an off-screen interface (which was the case in OT3 for both diagrams). Three of these participants were regular users of diagram editors. From these observations we conclude that considering the types of proxies to speed up navigation needs more training, but can be rather easily applied by experts.

Furthermore, we also rejected hypothesis **H4.** Combing the off-screen visualization technique with an overview window does not seem to improve orientation. During the study we observed that it is difficult for users to pay attention to both – overview and zoom+pan animation – at the same time during navigation. Nevertheless, participants appreciated the existence of an overview window, and it gave them the feeling of a better orientation. Corresponding to that, users of the *OS* interface wished to have an overview. From this we recommend that an overview window should be available. According to the findings of Nekrasovski et al. [27], it can serve as a *cognitive cushion* and can relieve users from mental load.

To improve orientation, we suggest visualizing the navigation path within the overview window. This can be combined with the history function presented in Section 7.3. When the user invokes this function, not only the proxies of the last visited nodes are highlighted, but in addition to that the navigation path is shown in the overview. In this way, users can observe the chronological order in which they have visited particular locations of the diagram. How this path is visualized in detail (e.g., by a path of arrows or highlighting respective locations by colors) is subject of further research.

Locate tasks. In our opinion the mean value of two attempts for finding the proper proxy for a given off-screen node is quite good for a rather short time of training. Two participants had problems finding the proper proxies for rather large off-screen classes. They stated that it was hard for them to estimate the location of the proxy according to the center of the class. Instead, they oriented themselves by the top or bottom border of the class and therefore chose the wrong proxy element.

8.4 Threats to Validity

For each controlled experiment threats to validity and limitations occur. For our study we see limitations in the UML and visual modeling experience of our participants, as they were no modeling practitioners. We reduced this threat by determining the participants' experience beforehand and created groups with a similar mean experience. Furthermore, we applied class diagrams with a limited amount of types of elements to reduce the complexity for inexperienced participants. The experiment was run with class diagrams only. Thus, the results are not generalizable for other UML diagrams or further diagram notations. Beyond that, we see threats to validity concerning the scalability to larger diagrams and the prior knowledge of the

participants about the given diagrams. These aspects are discussed in the following paragraphs.

Scalability. Concerning scalability, the study showed that our off-screen visualization technique works well for class diagrams with up to 100 nodes. We are confident, that our approach will also be beneficial for larger diagrams with several hundred nodes. Of course, a huge amount of nodes represented by a cluster leads to a long list of previews when the cluster is hovered. Exploring this list is time consuming and certainly increases the task completion times. However, it is hard to determine a concrete upper limitation concerning the number of nodes. If the off-screen technique is beneficial depends on several other factors, such as the given diagram layout, the density and connectivity of the diagram as well as the particular notation and given tasks. Moreover, in this study we did not consider some of the other proposed techniques which address scalability such as the *area of influence* (see Section 4.2.3) or interactive filtering (see Section 5.4). In which way these factors and techniques will influence the performance of the off-screen visualization should be carefully studied in the future.

Prior knowledge. Furthermore, in our study we confronted the participants with a diagram which was completely unknown to them. They were neither familiar with the content nor with the diagram layout and structure. In the future, we plan to run studies which cover other situations as well. Of course, known diagrams with familiar content can serve as datasets, such as class diagrams which were created manually by software modelers. Beyond that, we also plan to conduct evaluations with unknown diagrams (unknown layout) but familiar content. For that, class diagrams automatically generated from a known code base can be applied. We expect that in these cases participants will have fewer problems concerning orientation. Of course, these studies should be conducted with modeling experts.

Finally, further features and design alternatives can be added to the prototype and tested. Examples are the different representations of previews as described in Section 5.1, the techniques for showing further information within the border region as presented in Section 4.3 or the visualization of the navigation path within the overview.

9. CONCLUSION AND FUTURE WORK

In this paper, we contributed the application of off-screen visualization to the domain of node-link diagrams in general and to UML class diagrams in particular. In contrast to most of the off-screen techniques presented so far, our approach uses interactive proxy elements instead of simple graphical overlays to represent off-screen nodes. The proxies are visualized within a border region surrounding the viewport. This provides a contextual view of diagram elements usually not visible. Besides navigation by manual zooming and panning, our approach also supports automatic navigation by clicking on proxies. In that way, it is possible to navigate in a map-oriented way as well as based on the syntactical structure of the diagram.

We presented several approaches to make the change of edge routing as comprehensible as possible during panning and zooming. A preferable technique for that is *along edge projection* which does not affect the routing of edges at all. Furthermore, we presented ways to make our technique scalable to large diagrams with several hundred nodes. As solutions for that problem, we propose filtering and clustering of proxy elements (according to geometric and structural rules). Furthermore, if diagrams become larger we suggested a virtual *area of influence* around the viewport. It is utilized to filter nodes located further away.

The results of a pilot evaluation showed that the off-screen visualization technique is easy to understand and that creating proxy elements by radial projection towards the center of the viewport was preferred. In a second controlled experiment we found that for exploring relationships within unknown diagrams our approach outperforms state-of-the-art interfaces. Furthermore, participants were able to navigate to off-screen nodes without effort. We also found that the presence of an overview did not improve orientation within an unknown diagram. However, participants requested an overview as additional cognitive support.

For future work we will improve the performance of our prototype and add further functionality. In addition to the features described in the paper, the positioning of the proxies could be realized according to certain constraints by applying a mathematic optimization approach. In that way, the amount of geometric clusters could be minimized by translating proxies to the next free position, whereby the distance of a proxy to its original position is minimized as well.

Other aspects for future work are follow-up user studies involving modeling experts and using larger diagrams. Thereby, tasks should be used which consider the content and semantics of the visualized diagrams. As our approach is applicable to node-link diagrams in general, we will also apply it to other graphical notations, such as business process models, biological networks or feature trees used in feature-oriented software development [43]. In previous work we investigated techniques for diagram editing [12] and graph exploration [36] with multitouch and pen input on interactive surfaces. The prototype presented in this paper is integrated in the same system and also runs on multitouch enabled displays. Therefore, for future work we also plan to investigate how multitouch interaction techniques can be utilized for our off-screen visualization.

10. ACKNOWLEDGEMENTS

This work was funded by the German “Stifterverband für die Deutsche Wissenschaft” from funds of the Claussen-Simon-Endowment. Most parts of the work were realized at the Institute of Simulation and Graphics at the Otto-von-Guericke University Magdeburg, Germany. We thank Sebastian Kleinau, Ricardo Langner and Anne Rott for their great support. We also thank the anonymous reviewers for their insightful comments. Finally, we thank all the participants of the studies for their time and efforts.

11. REFERENCES

[1] Baudisch, P. and Rosenholtz, R. 2003. Halo: a technique for visualizing Offscreen objects. In *Proc. of CHI ‘03* (April 05 - 10, 2003), ACM, pp. 481-488.

[2] Bier, E. A., Stone, M. C., Pier. K., Buxton, W., and DeRose, T.D. 1993. Toolglass and magic lenses: the see-through interface. In *Proc. of SIGGRAPH* '93. ACM, pp. 73-80.

[3] Burigat, S., Chittaro,L. and Gabrielli, S. 2006. Visualizing locations of off-screen objects on mobile devices: a comparative evaluation of three approaches. In *Proc. of MobileHCI '06*. ACM, pp. 239-246.

[4] Burigat, S., Chittaro, L. 2011. Visualizing references to off-screen content on mobile devices: A comparison of Arrows, Wedge, and Overview + Detail, In *Interacting with Computers*, Volume 23, 2 (March 2011), pp. 156-166.

[5] Cherubini, M., Venolia, G., DeLine, R., and Ko, A. J. 2007. Let's go to the whiteboard: how and why software developers use drawings. In *Proc. of CHI '07* (April 28 - May 03), 2007, ACM, pp. 557-566.

[6] Cockburn, A., Karlson, A., and Bederson, B. B. 2008. A review of overview+detail, zooming, and focus+context interfaces. In *ACM Comp. Surv.* 41, 1 (Dec. 2008), pp. 1-31.

[7] Dobing, B. and Parsons, J. 2006. How UML is used. In *Commun. ACM* 49, 5 (May. 2006), pp. 109-113.

[8] Dwyer, T., Marriott, K., Schreiber, F., Stuckey, P., Woodward, M., and Wybrow, M. 2008. Exploration of Networks using overview+detail with Constraint-based cooperative layout. *IEEE Transact. on Visualization and Computer Graphics* 14, 6 (Nov. 2008), pp. 1293-1300.

[9] Eclipse UML, http://www.eclipse.org/uml2

[10] Eichelberger, H., Schmid, K., 2009. Guidelines on the aesthetic quality of UML class diagrams, *Information and Software Technology, Volume 51, Issue 12*, December 2009, pp. 1686-1698, ISSN 0950-5849.

[11] Frisch, M., & Dachselt, R., 2010. Off-screen visualization techniques for class diagrams. In Proc. of the 5th international symposium on Software visualization (pp. 163–172). New York, NY, USA: ACM.

[12] Frisch, M., Heydekorn, J. and Dachselt, R. 2010. Diagram editing on interactive displays using multi-touch and pen gestures. In *Proc. of the 6th international conference on Diagrammatic representation and inference* (Diagrams '10), Springer, pp.182-196.

[13] Frisch, M., Dachselt, R., & Brückmann, T., 2008. Towards seamless semantic zooming techniques for UML diagrams. In Proc. of the 4th ACM symposium on Software visualization (pp. 207–208). New York, NY, ACM.

[14] Furnas, G. W. 1986. Generalized fisheye views. *SIGCHI Bull.* 17, 4 (Apr. 1986), 16-23.

[15] Ghani, S., Riche, N. H., & Elmqvist, N., 2011. Dynamic Insets for Context-Aware Graph Navigation. Computer Graphics Forum, 30(3), 861–870.

[16] Gustafson, S., Baudisch, P., Gutwin, C., and Irani, P. 2008. Wedge: clutter-free visualization of Offscreen locations. In *Proc. of CHI '08* (April 05 - 10, 2008), ACM, 787-796.

[17] IBM Rational Rose, http://www.ibm.com/software/awdtools/developer/rose/

[18] Irani, P., Gutwin, C., and Yang, X. D. 2006. Improving selection of Offscreen targets with hopping. In *Proc. of CHI '06* (April 22 - 27, 2006), ACM, pp. 299-308.

[19] Irani, P., Gutwin, C., Partridge, G., & Nezhadasl, M., 2007. Techniques for interacting with off-screen content. In Proc. of the 11th IFIP TC 13 international conference on Human-computer interaction - Volume Part II (pp. 234–249). Berlin, Heidelberg: Springer-Verlag.

[20] Jacobs, T. and Musial, B. 2003. Interactive visual debugging with UML. In *Proc. of Symposium on Software Visualization* (June 11 - 13, 2003), ACM, pp. 115-122.

[21] Jusufi, I., Dingjie, Y. and Kerren, A. 2010. The Network Lens: Interactive Exploration of Multivariate Networks Using Visual Filtering, In *Proc. of 14th Conference Information Visualisation* (IV '10), IEEE, pp. 35 -42.

[22] Karnick, P., Cline, D., Jeschke, S., Razdan, A., & Wonka, P., 2010. Route Visualization Using Detail Lenses. IEEE Transactions on Visualization and Computer Graphics, 16(2), 235–247.

[23] Kagdi, H. and Maletic, J. I. 2007. Onion Graphs for Focus+Context Views of UML Class Diagrams. In *Proc. of VISSOFT '07*, pp. 80-87

[24] Microsoft Visio, http://office.microsoft.com/visio

[25] Moscovich, T., Chevalier, F., Henry, N., Pietriga, E., and Fekete, J. 2009. Topology-aware navigation in large networks. In *Proc. of CHI '09* (April 04 - 09, 2009), ACM, pp. 2319-2328.

[26] mxGraph, http://www.jgraph.com/

[27] Nekrasovski, D., Bodnar, A., McGrenere, J., Guimbretière, F., and Munzner, T. 2006. An evaluation of pan & zoom and rubber sheet navigation with and without an overview. In *Proc. of CHI '06* (April 22 - 27, 2006), ACM, pp. 11-20.

[28] Object Management Group, http://www.uml.org/

[29] Panagiotidis,A., Bosch, H., Koch, S., Ertl, T. 2011. EdgeAnalyzer: Exploratory Analysis through Advanced Edge Interaction, In *Proc. of the Hawaii Conference on System Sciences* 2011, pp. 1-10.

[30] Perlin, K. and Fox, D. 1993. Pad: an alternative approach to the computer interface. In *Proc. of* SIGGRAPH '93. ACM, USA, 57-64.

[31] Petre, M. 1995. Why looking isn't always seeing: readership skills and graphical programming. In *Commun. ACM* 38, 6 (Jun. 1995), pp. 33-44.

[32] Reinhard, T., Meier, S. and Glinz, M. 2007. An Improved Fisheye Zoom Algorithm for Visualizing and Editing Hierarchical Models. In *Proc. of the International Workshop on Requirements Engineering Visualization* (October 15 - 19, 2007). IEEE.

[33] Riel, A. 1996. Object-Oriented Design Heuristics. Addison Wesley, Boston MA, p. 32.

[34] Rohs, M. and Essl, G. 2006. Which one is better?: information navigation techniques for spatially aware handheld displays. In *Proc. of the 8th conference on Multimodal interfaces* (ICMI '06). pp. 100-107.

[35] Sarkar, M. and Brown, M. H. 1994. Graphical fisheye views. *Commun. ACM* 37, 12 (Dec. 1994), 73-83.

[36] Schmidt, S., Nacenta, M. A., Dachselt, R., and Carpendale, S. 2010. A set of multi-touch graph interaction techniques. In *Proc. of International Conference on Interactive Tabletops and Surfaces* (ITS '10), ACM, pp. 113-116.

[37] Sharp, R. and Rountev, A. 2005. Interactive Exploration of UML Sequence Diagrams. In *Proc. of VISSOFT '05* (September 25 - 25, 2005). IEEE Computer Society, Washington, DC, p. 8.

[38] Shneiderman, B. 1996. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proc. of the IEEE Symposium on Visual Languages* (VL '96). IEEE Computer Society, Washington, p. 336

[39] Soukup, J. and Soukup, M. 2007. The Inevitable Cycle: Graphical Tools and Programming Paradigms. *Computer* 40, 8 (Aug. 2007), 24-30

[40] Sparx Systems, http://www.sparxsystems.com/

[41] Spence, R, Apperley, M. 1982. Database navigation: An office enironment for the professional. *Behav.Inf. Technol. 1*, 1, pp. 43–54.

[42] A. S. Spritzer and C. M. D. S. Freitas, "Design and Evaluation of MagnetViz - A Graph Visualization Tool," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, pp. 822-835, 2012.

[43] Stengel, M., Frisch, M., Apel, S., Feigenspan, J., Kästner, C., & Dachselt, R. 2011. View infinity: a zoomable interface for feature-oriented software development. In *Proc. of ICSE 2011* (pp. 1031–1033). New York, NY, USA, ACM.

[44] Tominski, C., Abello, J., van Ham, F., and Schumann, H. 2006. Fisheye Tree Views and Lenses for Graph Visualization. In *Proc. of the Conference on information Visualization* (July 05 - 07, 2006). IEEE Computer Society, Washington, DC, pp. 17-24.

[45] Tominski, C.; Abello, J.; Schumann, H. 2009. Two Novel Techniques for Interactive Navigation of Graph Layouts, In *Proc of EuroVis'09*, Berlin.

[46] Turetken, O., Schuff, D., Sharda, R., and Ow, T. T. 2004. Supporting systems analysis and design through fisheye views. *Commun. ACM* 47, 9 (Sep. 2004), pp. 72-77.

[47] van Wijk, J., Nuij, W. 2004. A Model for Smooth Viewing and Navigation of Large 2D Information Spaces, In *IEEE Transact. on Visualization and Computer Graphics*, pp. 447-458.

[48] Wong, N., Carpendale, S. and Greenberg, S. 2003 EdgeLens: An Interactive Method for Managing Edge Congestion in Graphs**.** In *Proc. of InfoVis 2003*. IEEE Press, pp. 51-58.

[49] Wu, J., and M.-A. Storey. 2000. A multi-perspective software visualization environment, In *Proc. of CASCON'00*, November 2000, pp. 41-50.

[50] Wybrow, M., Marriott, K., Stuckey, P.J. 2006. Incremental connector routing. In: GD 2005. Volume 3843 of LNCS., Springer, pp. 446-457.

[51] Zellweger, P. T., Mackinlay, J. D., Good, L., Stefik, M., and Baudisch, P. 2003. City lights: contextual views in minimal space. In *CHI '03 Ext. Abs. on Human Factors in Computing Systems* (April 05 - 10, 2003). ACM, pp. 838-839.

[52] Zest, Eclipse Visualization Toolkit, http://www.eclipse.org/gef/zest/

Appendix

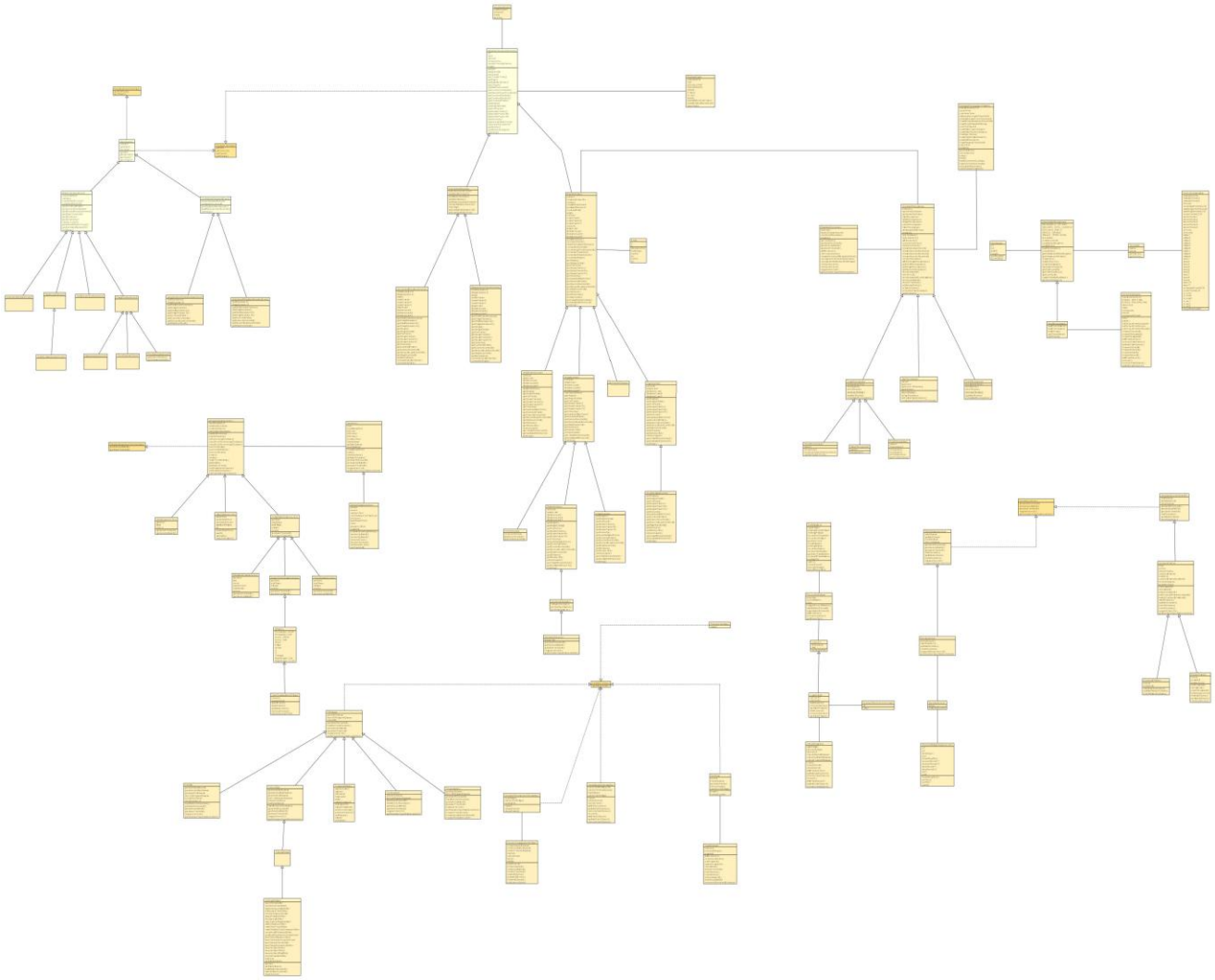

Figure 19. Diagram D1 which was used for the comprehension, orientation and locate tasks.

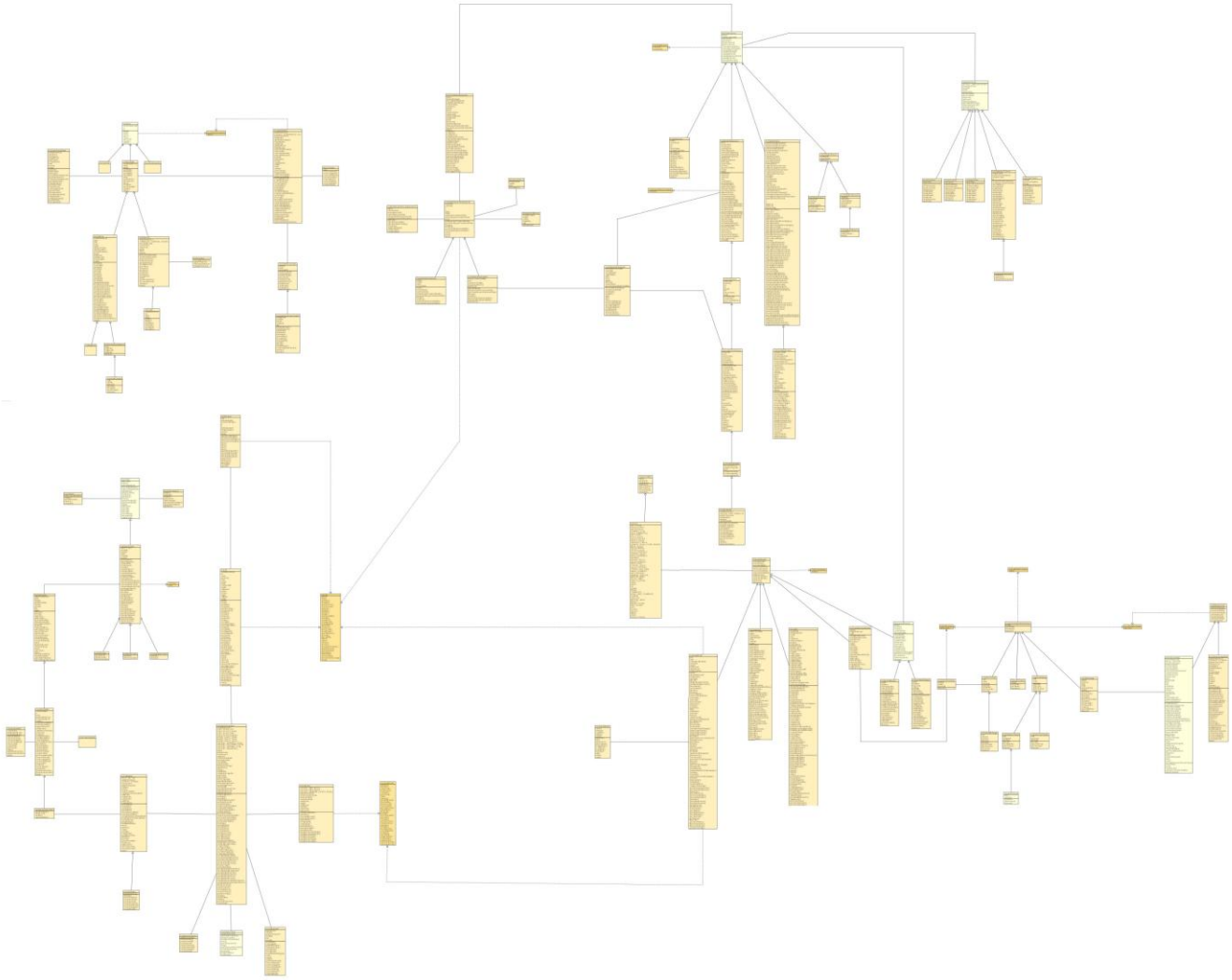

Figure 20. Diagram D2 which was used for the comprehension, orientation and locate tasks.