Off-Screen Visualization Techniques for Class Diagrams

Mathias Frisch, Raimund Dachselt
User Interface & Software Engineering Group
Otto-von-Guericke University
Magdeburg, Germany
[mfrisch, dachselt]@isg.cs.uni-magdeburg.de

ABSTRACT

Visual representations of node-link diagrams are very important for the software development process. In many situations large diagrams – probably consisting of hundreds of nodes and edges – have to be edited and explored. In state-of-the-art modeling tools these activities are often accompanied by time consuming panning and zooming. 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 the approach is to give a contextual view of all nodes which are clipped from the current viewport. Nodes are represented by proxy elements located within an interactive border region. The proxies show information of the associated off-screen nodes and can be used to quickly navigate to the respective node. However, there are several challenges when this technique is adapted to node-link diagrams, for example concerning the change of edge routing or scalability. We describe the design space of this approach and present different visualization and interaction techniques in detail. Furthermore, we conducted a formative evaluation of our first prototype. Based on the observations made during the evaluation, we came to final suggestions how particular techniques should be combined.

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 [2]. Over the last 15 years the Unified Modeling Language (UML) [17] 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 [4, 21] 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 users orient themselves in a "geographic way" similar to map navigation. This means that they know for example the spatial location or the direction of a particular node. On the other hand navigation can be performed by means of the diagram topology and diagram semantics. For example, users often know which nodes are connected or on which level of a tree structure a particular node is located. Contextual semantic information is important for this kind of navigation. For example, properties of edges, such as labels or

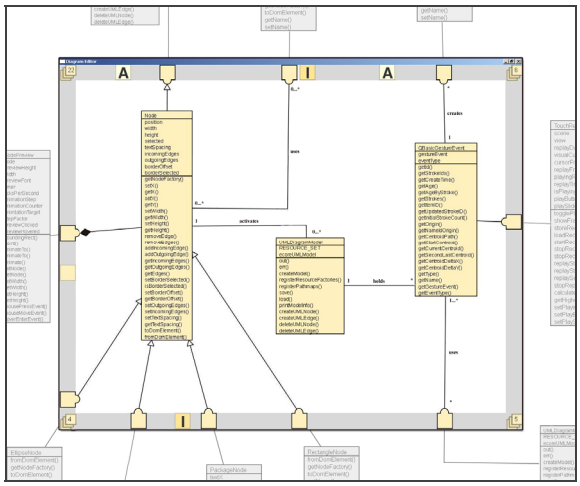

Figure 1. Viewport of UML class diagram editor with off-screen visualization (center). Classes clipped from the view-port (shown outside in gray) are represented by proxy elements located within the interactive border region.

SOFTVIS'10, October 25–26, 2010, Salt Lake City, Utah, USA.

multiplicities, and types of connected nodes must be available. Usually, state-of-the-art modeling tools only support the map navigation approach and offer zooming and panning combined with overview and detailed view. 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. They are not visible anymore and can only be reached by means of 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. Furthermore, important contextual information is also invisible if these traditional techniques are applied. In order to overcome this problem, focus+context techniques have been investigated. They usually distort the content of the context region, e.g. by means of a degree of interest function (DOI) [8] or in a geometric way [19]. However, they also give little or no information on a semantic level.

In this paper we investigate off-screen visualization techniques for node-link diagrams as an alternative to traditional overview+detail or distortion oriented focus+context techniques. Up to now, off-screen visualization techniques were mainly applied to mobile devices [1, 9]. 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 display (see Figure 1). Furthermore, they serve as links providing automatic navigation to the associated off-screen node. Proxy elements offer spatial information as well as semantic information about elements currently clipped. In that way, our technique supports both, map oriented navigation and navigation based on diagram semantics.

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 semantic rules. We implemented a prototype for navigating and editing a selected subset of UML class diagrams. This application was used to conduct a formative evaluation. Observations and comments collected during the study led to final suggestions which concrete techniques of the design space should be combined.

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 expert evaluation is described in Section 7. 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. [3]. 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 [10, 14, 22]. 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 in order 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. [5] 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. Sharp et al. [20] 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 there is more cognitive load, as users have to switch between both views [3]. Beyond that, Nekrasovski et al. [16] 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. [26] and Moskovich et al. [15] 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. In order 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. it 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 [8] or context elements are geometrically distorted [19]. 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. [19]. 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. [27] and Reinhard et al. [24] seize on this approach and apply it in order to visualize hierarchical nesting of nodes. Particular nodes, e.g. of business process models and data flow diagrams [27], can be expanded in order to show nested nodes of a finer level. This technique is also applied in ShriMP [29]. Besides fisheye techniques, ShriMP offers also semantic zooming and multi-focus visualization. It has been applied to visualize the structure of ontologies and Java programs, e.g. by means of call graphs. Jacobs et al. [12] use a fisheye technique is 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. [13] 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

Local distortion is often applied by means of lenses. For example, Tominski et al. [25] presented different lenses for graph visualization. The approach can be used e.g., to bring connected neighbors of a selected node towards the focused area. A similar 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, Tominski et al. [26] developed a radar view mode for graphs. During navigating a graph 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

In contrast to the aforementioned approaches, cue-based techniques do not distort or modify the elements located in context. Rather, proxies for the elements which are located in the off-screen area are created. These proxies are often shown as overlays at the border of the display. 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 [9]. The latter were mainly developed for map navigation on small displays of mobile devices. They are designed to indicate the existence, the direction of and the distance to off-screen elements by means of overlays. However, they do not show further information about the off-screen element such as its type, and they are not interactive.

City Lights [31] 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. [11] 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.

The study conducted by Nekrasovski et al. [16] compared zoom+pan with focus+context (a rectangular rubber sheet) for navigation tasks within a large binary tree. Results showed that the zoom+pan interface was faster and demanded less mental effort than the focus+context interface. Beyond that, Halos were used to indicate the position of already visited nodes. 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. We go beyond this rather simple adaption of already existing approaches and contribute techniques such as clustering strategies for proxy elements e.g., based on diagram semantics, two different ways of projecting off-screen nodes and visualizing a variety of additional semantic information.

3. THE OFF-SCREEN VISUALIZATION APPROACH

In order to make diagram editing and navigation tasks more efficient and effective, we seize on the off-screen approaches discussed in 2.3. We contribute their application 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 application 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 means of 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. [31] 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. As mentioned above, we indicate the existence of off-screen nodes by means of proxy elements. Proxies are created by projecting the position of the clipped nodes to the border of the currently visible part of the workspace. 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, diagram items hold distinct informational properties. There are, for example different types of nodes. For UML class diagrams we currently distinguish classes, abstract classes and interfaces. These properties are mapped to the color and the labeling of proxy elements to show the type of the associated node (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. 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 in order to support manual navigation. 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 in order to navigate to the respective off-screen node. This technique allows 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 [9], 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 semantic information such as the type of a clipped node.

Interaction. 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. The technique should be applicable for large diagrams with at least hundreds of nodes. However, off-screen visualization techniques usually suffer from cluttered proxies if a large amount of off-screen elements exist. We try to overcome this problem by automatic clustering and interactive filtering of proxy elements. Different clustering strategies are presented in Section 4.2 and filtering is presented in Section 5.3.

Shape of proxies. Indicators such as arrows, halos or wedges are hard to distinguish from edges and their visual properties

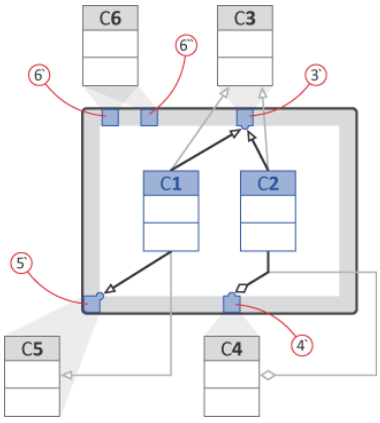

Figure 2. Proxy elements are created by projecting off-screen classes onto the interactive border region (gray area). Class CX is represented by proxy X'. For edges connected with proxy elements the routing is changed (see aggregation between C2 and C4).

(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 squared shape.

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 [18] 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 (in order 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. [7]. 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 order to fulfill the requirements and master the challenges mentioned in the previous section, we investigated several design alternatives for all parts of our visualization technique. In this section 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. In Figure 2 Class C1 and C2 are both on-screen and connected with Class C3 by means of 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.

In Subsection 4.1.1 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

Basically, projecting nodes onto the border of the viewport can be done in two ways: either *orthogonal* or *radial*. Both ways clearly indicate the direction of an off-screen node. We subsume these two possibilities as geometric projection. For orthogonal projection nodes are projected perpendicular to the border of the viewport. For radial projection the center of the projection is located in the center of the viewport. 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.

However, 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 particularly, this becomes problematic if an edge is bent and consists of several segments. This can be observed for 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 segment* 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. *Proxy edge segments* can be rendered in a different color than actual edge segments in order to signal that they do not represent the original edge (see Figure 2 and Figure 4 where *proxy edge segments* 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 [7] can be violated, as edges crossing each other or edges crossing nodes can occur. In the following subsections 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. In order to deal with 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 edge segment*s 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 display. With this approach the visible part of an edge maintains its routing completely. *Proxy edge segments* start at the intersection point of the edge and the border of the display 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 segment* is rendered in a rubber band style e.g., by means of a Bezier curve. The general drawback of this solution is that edgeclutter can occur along the border of the viewport if an off-screen node has many edges.

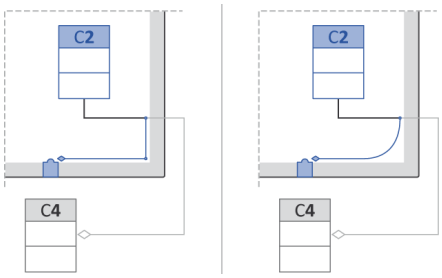

Figure 3. Concept sketch for routing edges along the border of the viewport: rerouting by straight proxy segments (left) and by rubber band (right).

4.1.2 Projection along Edges

In order 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 Figure 2 but with *along edge projection*. Off-screen nodes 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'). In order to distinguish the way a node was projected, we suggest applying two types of border colors for proxy elements. Proxy elements projected along edges have a darker border color than proxies created by geometrical projection (see Figure 4). Beyond that, they are rendered always in the foreground and are never aggregated in geometric clusters (see 4.2.1). 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 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.

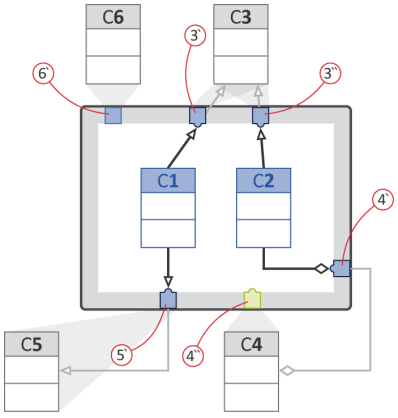

Figure 4. Proxy elements are created by means of *along edge projection.* For class C3 two proxies are created (3' and 3''), if proxy 4' is hovered, 4'' appears to indicate the proper direction of C4.

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).

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 orange 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".

In order to connect edges with proxy elements, each proxy owns a so-called edge port. An edge port is a semicircular extension of a proxy element. It has the same color and reaches from the interactive border region into the workspace. If an off-screen node is connected with several visible nodes the respective edges are attached at the edge port in order to prevent clutter of edges. This is necessary, especially when properties such as arrow heads are present (see Figure 5 right). Edge ports only appear when the associated off-screen node is connected with visible nodes.

In order 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 particular, there are two different ways of clustering proxy elements: geometric and semantic clustering. Both can be applied simultaneously.

4.2.1 Geometric Clustering

Geometric clustering is applied if more than one node is projected to the same position of the interactive border region. In that case, a cluster proxy is created. For an example see Figure 6 (case 1, 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. Whether this *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

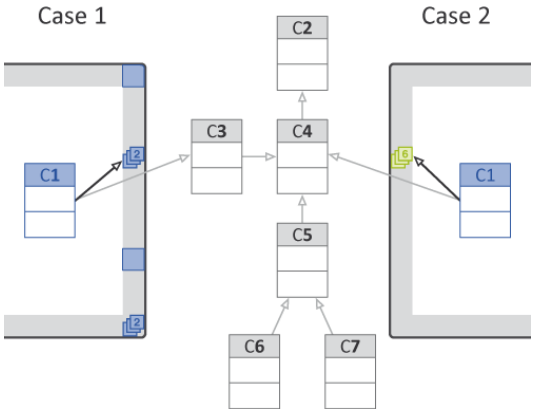

Figure 6. Geometric clustering (case 1, left) and semantic clustering of an inheritance hierarchy (case 2, right).

beneficial. For these kinds of diagrams arranging nodes in a vertical or horizontal layout is part of the secondary notation [18].For example, placing proxy elements above each other, although their associated nodes are arranged in a horizontal line, can be confusing here.

4.2.2 Semantic Clustering

Besides geometric clustering, proxy elements can also be clustered according to semantic rules based on 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 (case 2, 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, semantic 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. Semantic 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.

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. For a 1D-border proxy elements can be drawn as symbols with different colors, shapes or labels. In particular, approaches such as the onion-graph notation [13] can be applied for clustered inheritance hierarchies in class diagrams. 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 [23] providing a condensed view of the remaining diagram within the border. Furthermore, we propose to use

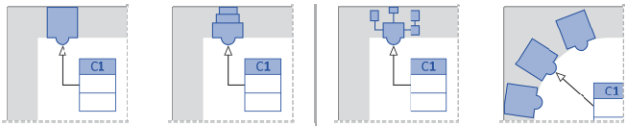

Figure 7. Different dimensions of the border region, from left to right: 1D, 1.5D and 2D. Border region with rounded corners (right).

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.

5. INTERACTION

The positions of proxy elements are constantly updated during manual panning and zooming. 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, in order to make the relation of node and proxy comprehensible.

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. 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. Furthermore, we suggest that previews can be expanded to show the content of the respective node.

Besides that, if a visible node is selected, the proxy elements which are directly connected with the selected node are highlighted, and all their previews are shown. In that way, a user can easily get more information about nodes the currently selected one is connected with.

5.1 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. In order to make the automatic navigation as smooth and comprehensible as possible, we applied simultaneous panning and zooming according to van Wijk and Nuij [28].

If a cluster proxy is clicked, the viewport is animated in a way that all clustered nodes are focused. In order 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 semantic clusters all associated nodes are projected by means of geometric projection resulting in proxy elements at the respective location.

5.2 Inserting Edges

Besides providing a quick navigation to clipped nodes and guaranteed visibility of edge properties, we also support creating edges between visible nodes and off-screen nodes. Edges can be dragged to proxy elements of the border region and are 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. 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. [30] should be applied.

5.3 Interactive Filtering

In addition to automatic clustering we propose interactive filtering of proxy elements in order to prevent clutter and to make our technique scalable to large diagrams. Filter criteria can be adjusted interactively by means of the graphical user interface. 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 with a huge amount of attributes and methods are shown).

6. IMPLEMENTATION

We implemented the off-screen visualization approach as a prototype for navigating and editing UML class diagrams. The application is written in Java whereby the graphical user interface is based on Qt Jambi. The prototype is based on the Eclipse UML model [6] and diagrams can be imported by means of XMI. The layout of a diagram is stored in a separate file, also using an XML format.

Figure 8 shows two screenshots of the prototype. The class diagram is shown in the center region with white background. Proxy elements for off-screen nodes are placed within the interactive border region with light gray background. 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 first prototype is capable to visualize class diagrams consisting of classes, abstract classes and interfaces. Relationships are limited to associations and generalizations. 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

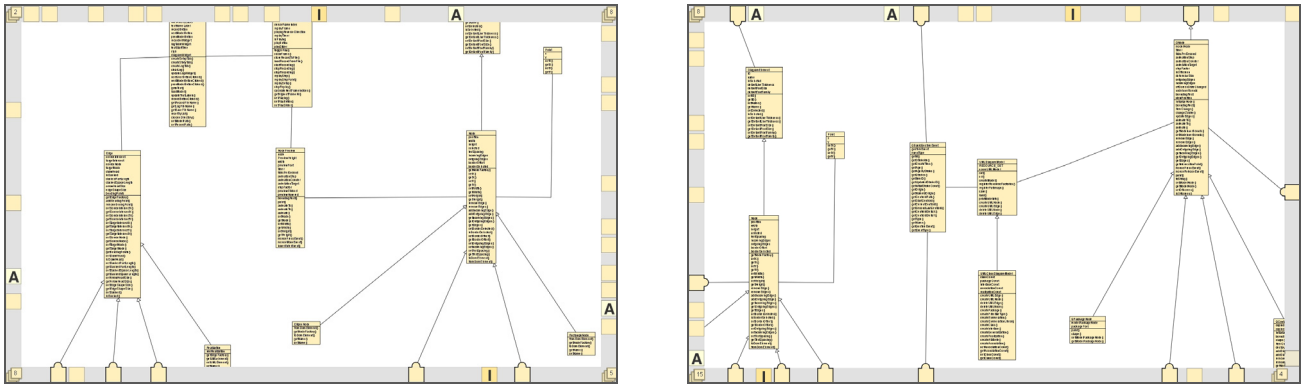

Figure 8. Two screenshots of our prototype. A particular part of the class diagram is focused (left). Nodes located off-screen are represented by proxies within the interactive border region. The position of the proxies is dynamically updated during panning and zooming. For example, the screenshot at the right hand side shows the result of panning the left view to the left.

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 the aforementioned algorithm for avoiding clusters (see Section 4.2). If there is enough space available proxies are placed side by side until a certain distance threshold is reached. Besides that, we implemented semantic 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).

7. EVALUATION

We conducted an evaluation of our early prototype. Our goal was to collect feedback at an early stage of development, in order to come to decisions for further design iterations. In particular 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?

7.1 Design of the Evaluation

We conducted the evaluation in a formative way and applied a think-aloud approach in combination with user observations and a questionnaire.

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’’.

Participants. Eight participants (6 male, 2 female, age from 24to 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 an example diagram. 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. For the evaluation orthogonal projection for unconnected nodes and *along edge projection* for connected nodes was used.

The evaluation procedure was structured in two parts. Part one consisted of a guided navigation within a smaller class diagram. During the procedure, we asked the participants to perform particular tasks and about their opinions concerning certain design issues. 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. The diagram consisted of 31 classes (3 of them abstract) and 35 relationships (18 associations and 17 generalizations). In order 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 means of an association. Furthermore, a stage play is a special kind of *event* – expressed by means of a generalization. The diagram was layouted manually according to aesthetic rules [7]. 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.

During the guided navigation we asked the participants to perform several smaller tasks. For example, 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 or to navigate to a certain class and tell its directly connected neighbors. Furthermore, we asked them to count abstract classes in order 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. In order 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 observations, comments and suggestions of the participants. Beyond that, at the end we handed a questionnaire to them with five 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 "agree" to "completely disagree").

7.2 Results and Discussion

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.

In general, the automatic zoom and pan navigation was comprehensible. However, some participants commented that it was too quick and should zoom out more during panning to give a decent overview. This 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. 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 (e.g. by arrows).

Appearance of Proxies. Proxy elements representing classes directly connected with visible nodes were clearly distinguishable from other proxy elements. 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, 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. 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 will be carefully studied in the future.

All these observations considered we come to the following final suggestions: proxies not connected with visible nodes should be created by means of radial projection as most of the participants expected radial projection. Furthermore, many participants were confused when a node with many edges was represented by several proxies due to *along edge projection*. In order to mitigate this problem, we propose a slight adoption of the approach applied in the evaluation. Both projection techniques – geometric and *along edge projection* – should be applied simultaneously. Geometric projection should be used for nodes connected with straight edges. In this case the change of edge routing is rather easy to comprehend, and it is ensured that there is exactly one proxy for the node. Furthermore, *along edge projection* should only be applied for nodes connected with edges which are bent several times and not completely visible in order to prevent confusing change of edge routing.

8. CONCLUSION AND FUTURE WORK

We contributed the application of off-screen visualization to the domain of node-link diagrams in general and to UML class diagrams in particular. Thereby, clipped nodes are represented by proxy elements within an interactive border region surrounding the viewport. Proxies provide a contextual view of information usually not visible such as the type of the associated node and the type of connected edges. Our approach supports map-oriented navigation as well as navigation based on the diagram semantics. In particular, it realizes automatic navigation to arbitrary off-screen nodes.

We investigated the design space for our approach concerning visualization and interaction techniques. Thereby, we contributed solutions to challenges such as the change of edge routing during panning and zooming and the scalability to large diagrams. The visualization technique was implemented as a first prototype for UML class diagrams. An evaluation showed promising results and led to final suggestions concerning the projection of nodes and handling the change of edge routing.

For future work, we will adopt our prototype based on the results of the expert evaluation. Furthermore, we plan to conduct further user studies. In particular, we will investigate different dimensions of the border region and the combination with

overview+detail techniques. As our approach is applicable to node-link diagrams in general, we will also apply it to other notations such as business process models.

9. ACKNOWLEDGEMENTS

This work was funded by the "Stifterverband für die Deutsche Wissenschaft" from funds of the Claussen-Simon-Endowment. We thank Sebastian Kleinau, Ricardo Langner and Anne Rott for their great support.

10. REFERENCES

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

[2] 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* (San Jose, California, USA, April 28 - May 03, 2007). ACM, pp. 557-566.

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

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

[5] 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.

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

[7] 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

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

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

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

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

[12] Jacobs, T. and Musial, B. 2003. Interactive visual debugging with UML. In *Proc. of Symposium on Software Visualization* (San Diego, California, June 11 - 13) SoftVis '03. ACM, pp. 115-122.

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

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

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

[16] 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* (Montréal, Québec, Canada, April 22 - 27, 2006). ACM, pp. 11-20.

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

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

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

[20] 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, 8.

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

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

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

[24] 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). REV. IEEE Computer Society

[25] 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

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

[27] 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.

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

[29] Wu, J., and M.-A. Storey. 2000. "A multi-perspective software visualization environment", in *Proceedings of CASCON'2000*, November 2000, pp. 41-50.

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

[31] 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* (Ft. Lauderdale, Florida, USA, April 05 - 10, 2003). ACM, pp. 838-839.