ZOOMABLE USER INTERFACES IN SCALABLE VECTOR GRAPHICS

Martin Rotard, Mike Eissele, Raoul van Putten, Thomas Ertl
Universität Stuttgart
Visualization and Interactive Systems Institute
Universitaetsstrasse 38
70569 Stuttgart
{rotard, eissele, ertl}@vis.uni-stuttgart.de, rvputten@gmail.com

ABSTRACT

Zoomable user interfaces are an evolutionary outgrowth of graphical user interfaces. In this paper we propose a zoomable user interface based on Scalable Vector Graphics. Three-level zooming is proposed as a new paradigm to combine different zooming functionalities in a common interface and support zooming within the window manager. This helps to unify zooming techniques of different applications. To meet the demand of efficient and easy navigation on a user interface, several novel interaction techniques are shown that further support the integration of three-level zooming within the underlying presentation system. For mobile small-screen devices, where the benefit of zooming user interfaces is even higher, the proposed system can be operated with simple *pen tap* or *tap and drag* operations. We also present a prototypical implementation, which demonstrates how applications based on the SPARK toolkit can transparently benefit form the proposed technology.

KEYWORDS

Zoomable User Interface, Graphical User Interface, Interaction, Human Computer Interaction, Scalable Vector Graphics

1. INTRODUCTION

A zoomable user interface is a successor to the traditional windowing graphical user interface. This modern interface paradigm solves the problem of the limited screen real estate and enables the user to have windows, documents, and folders in any position on the desktop and in a special presentation size. Humans tend to remember landmarks and relative positions. This behavior can be adapted for zoomable user interfaces where information is accessed on the desktop by zooming and panning to a specific view. Stepping back to get an overview of the area of interest or over the desktop is given by zooming out. Particularly with regard to small displays like mobile devices have, zoomable user interfaces are an advantage for the user in handling several applications. Zoomable user interfaces are a key feature to deploy advanced applications and desktops on small mobile devices. In general, there is no restriction in the usage of this user interfaces paradigm and, therefore, also small-screen devices and even huge high-resolution powerwalls benefit. To be flexible in the area of application we focus in this paper on presentation and navigation metaphors that have minimal system requirements. Thus our proposed concepts can be used with limited screen space and can even be controlled with just a single-button pen interface.

There is no rule that defines which functionalities – except zooming and panning – a zoomable user interface needs to have. The combination of zoomable user interfaces with a concept of *level of detail* seems to be very promising. The result will be a desktop where document icons are a preview of the document that will increase in detail when the user zooms in and folders are opened by zooming on them. Another vision is the combination of zoomable user interfaces with *focus+context* techniques. This will enable users to zoom, e.g., on an image, embedded in a text document. In a defined zoom level the image will directly be editable within another document and the textual information of the context will still be visible. So users can zoom out and edit the entire document and zoom in to focus on editing a part of the document in a predefined

application. This will help the users to handle the still increasing number of different applications that are used simultaneously.

The realization of the zoomable user interface paradigm needs to have a scalable graphics technology as basis. Without the mathematical description of widgets a zoomable user interface will result in an unattractive pixelized mosaic. Scalable Vector Graphics (SVG) can be used as an underlying technology, as it supports a mathematical description of shapes and positions. In the recent years, several groups have developed widget sets using SVG, which can be used to realize a zoomable user interface. For the prototypical implementation we chose the SPARK (SVG Programmers' Application Resource Kit) Toolkit and extended it to support zooming. Therewith, applications utilizing SPARK are naturally extended with a zoomable user interface. Several paradigms and interaction techniques are shown that support an efficient handling and navigation on zoomable user interfaces.

The rest of this paper is structured as follows: The next section describes related work in the area of zoomable user interfaces. In section 3 zooming on the desktop is outlined. Section 4 shows navigation metaphors in zoomable user interfaces like three-level zooming and zoom bars. Our prototypical implementation is described in section 5. The paper ends with a discussion and gives an outlook to future work.

2. RELATED WORK

The concept of zoomable user interfaces has been examined in several projects in the last decade. Zoomable methods have been used in Pad respectively Pad++ [Perlin et al. 1993, Bederson et al. 1994]. The experience obtained in the development of these projects was used to build the toolkits Jazz and Piccolo [Bederson et al. 2000, Bederson et al. 2004]. Especially Piccolo is very flexible in its usage as it is a layer built on top of a lower level graphics API for Java and .NET.

Jef Raskin described metaphors for navigation in and usage of zoomable user interfaces [Raskin 2000]. His ideas have been turned into the *Archy* project (also used to be called *The Humane Environment*) [Archy] that is worked on by the Raskin Center for Humane Interfaces. A central concept in this system is the user interface *Zoomworld*. Sun Microsystems supports the free software project *Looking Glass* where a 3D desktop environment for Linux and Solaris is developed [Looking Glass]. There are several studies that compare the concept of zoomable user interfaces with common methods of user interfaces [Hornbæk et al. 2002, Gutwin et al. 2004, Hightower et al. 1998].

The graphical subsystem *Windows Presentation Foundation* in *Microsoft Windows Vista* uses the *eXtensible Application Markup Language* (XAML) for the user interface. This enables the system to integrate zoomable user interfaces. *Upper Bounds Interactive Inc.* with their product *Tactile 3D* present a three-dimensional interface to manage files and folders [Tactile 3D]. The system allows users to freely navigate in a 3D environment where the installed files and programs are distributed. For concentrating on a specific file, users can enter a dedicated area. However, navigating in 3D is no easy task, especially when entering different areas in which the user cannot get the overall overview.

In the recent years widget libraries for SVG have been defined. For our zoomable user interface we have used the *SVG Programmers' Application Resource Kit* (SPARK) [Lewis et al. 2003, SPARK] because of its Model-View-Controller architecture within each component. This makes it possible to zoom each widget independently. *Dynamic Scalable Vector Graphics* (dSVG) and *SVG Widget New Edition* (SWiNE) are other widget libraries for SVG that could be extended by a zooming metaphor [dSVG, SWiNE].

3. ZOOMING ON THE DESKTOP

As shown in the previous section, zoomable working environments have been researched in many projects, most of them utilized zooming to simply scale the desktop in order to show more or less detail. Further more, many applications support different kinds of zooming, e.g., image editors, text editors, or document viewers. This shows that zooming is an often applied functionality, but each application makes use of it in a slightly different kind. For that purpose, users have to learn and understand the different kinds of zooming and their corresponding interaction technique to make use of them.

In contrast, this paper proposes paradigms to further improve zoomable user interfaces and unify the zooming techniques of different applications to a general concept for zooming; named: *Three-Level Zooming*. Further more, *Application Level-Of-Detail Zooming* is presented to transparently switch to applications, which are used to generate specific content within contextual documents or other applications.

3.1 Three-Level Zooming

The goal of three-level zooming is to support zooming within the underlying window management in a way that applications do not need to support zooming, as they automatically make use of the zooming functionality of the system. Three semantically different levels for zooming have been identified for a zoomable user interface:

Desktop Level – Zooming the desktop environment is used to choose the level of detail over the entire working space. In general, this basic level is supported by nearly all proposed zoomable user interfaces. Through this functionality users can easily get an overview of the entire desktop by zooming out in order to view all nearby windows simultaneously. At this level the windows may only show a rough approximation of their content, however, still enough for a user to recognize all active applications and their corresponding window. To get more detail of a specific area, the user can simply zoom into the area of interest. Thereby, the context information of all opened windows diminishes continuously until only a single application window is shown on the entire screen. An example of an overview of the entire working space of a zoomable user interface is illustrated in Figure 1.

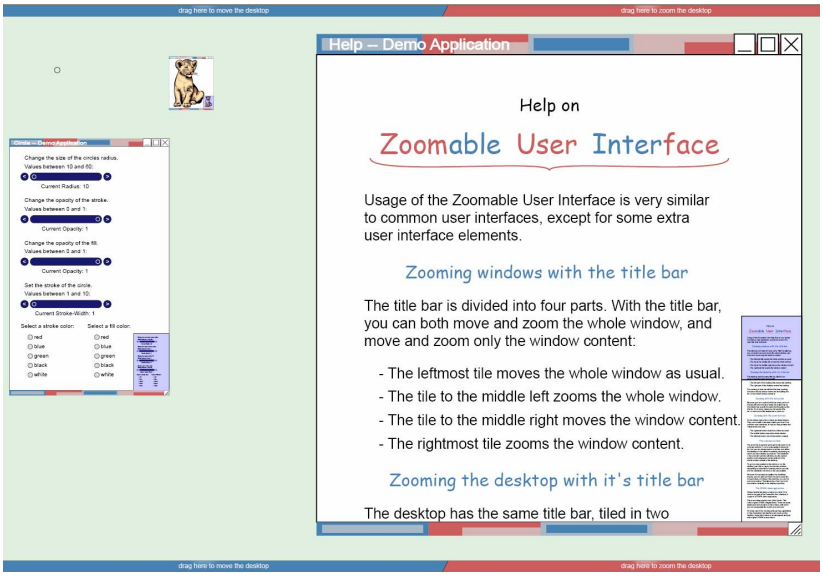

Figure 1: Overview of the entire working area

Application Window Level – Resizing the entire application window including menu bar, short-cut icons, and content. Often, applications provide a user interface via menus, short-cut icons, buttons, etc. The more powerful an application is, the more advanced and complicated its user interface gets. In contrast, simple applications with a limited functionality mostly provide only a simple user interface with, e.g., only a single button. On today's windowing systems users are forced to rely on the application programmers to choose an adequate size for the interaction elements, e.g., for buttons. On a zooming user interface the display size on which the application is operated has a much higher variety than for fixed sized user interfaces. This fact makes it hard for programmers to choose an adequate user interface design. In addition, experienced users memorize the user interface layout and are able to operate applications at a reduced display size for the user interface – e.g. smaller buttons without a caption – in order to save display size for important content. With zooming at the *application window level* users can freely control the display size of the entire application including the user interface such as menu bars, buttons, checkboxes, etc. Figure 2 depicts a desktop with three applications at different zooming levels. The applications with simple user interfaces are scaled down in order to free desktop space for the more advanced application.

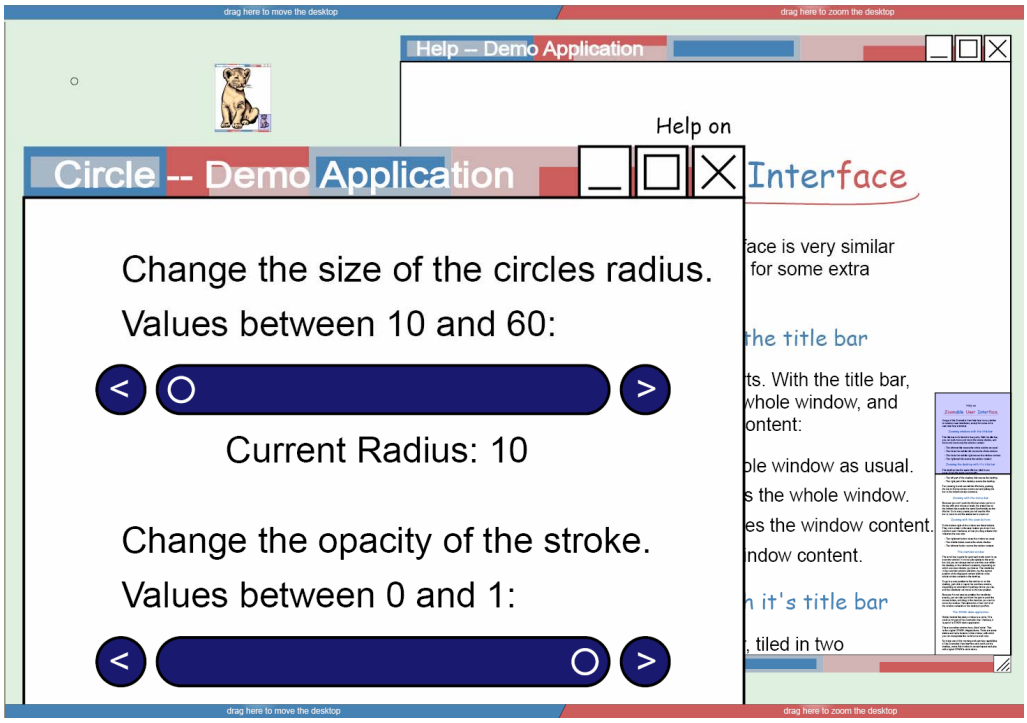

Figure 2: Application window level zooming for working at different levels of detail

Application Content Level – Resizing the content of an application window. There are numerous applications that support some kind of zooming. Most of them support zooming of the content, detailed view for editing and an overview mode for manipulating the layout. This kind of zooming can be found in, e.g., word processors, painting programs, or modeling tools. Unfortunately, this functionality is not standardized and each application provides a different look&feel for it. For programs running on a zoomable user interface this problem is even more severe, as zooming the content of the application is used more frequently. Therefore, application content level zooming provides a novel technique to support zooming of the application's content by the underlying windowing system in an application independent way. Figure 3 shows two application windows at different zoom settings in the application content level.

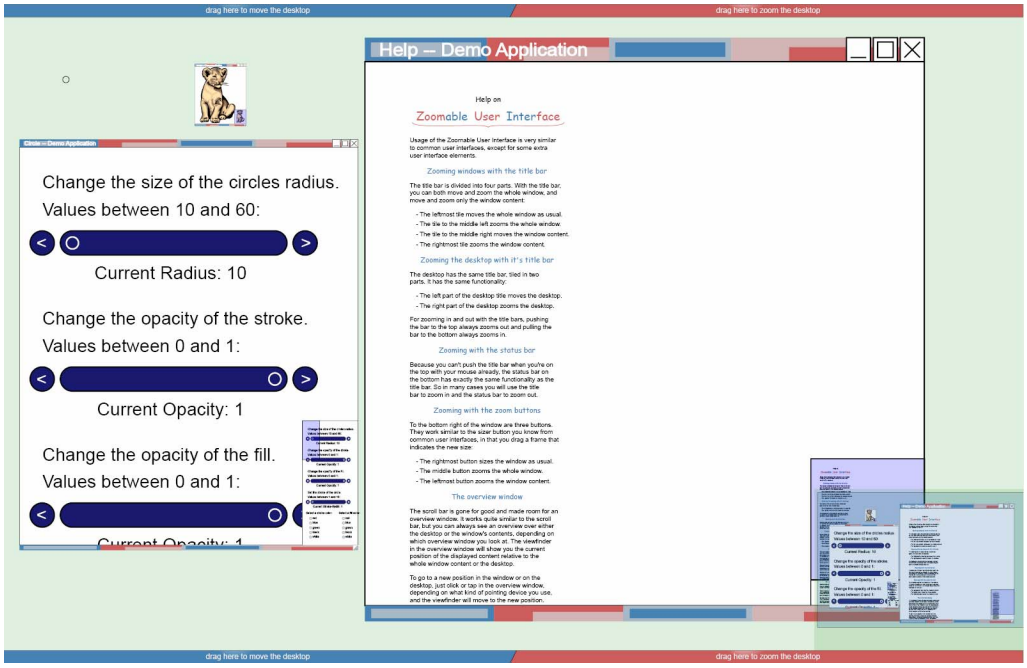

Figure 3: Two application windows with different zoom settings at the application content level

Supporting all three above mentioned zooming techniques enables users to freely organize their windows – in position, size, and level of detail – within the entire working area, independent of the application. In combination with efficient navigation techniques to control the different zooming levels, this greatly supports the users in using multiple applications concurrently to perform their tasks.

3.2 Application Level-Of-Detail Zooming

As stated earlier, the number of different applications concurrently used by users working on computers will further increase with the utmost probability in future. Each application is designed to handle specific tasks where the output can be used as content within other applications. A major task for window managers already is and will be to efficiently support working with multiple applications to generate content for other applications. Most of the research in this field has been done in the infrastructure technology, reaching form Object Linking and Embedding (OLE) to Compound Documents, where content of different applications are combined within a single document. In contrast, working with these technologies has not been a focus of research and can be greatly enhanced via a zooming user interface. The *application content level* zooming – described in the previous section – can be extended to handle *application level-of-detail* zooming in a way that a user smoothly switches between applications via zooming content.

Todays compound documents or OLE technologies already allow editing particular content objects via dedicated applications within the surrounding document. However, this is often not practical as the working space and user interface of the embedded application is too small and users therefore tend to decouple the application from the surrounding document to enlarge the working area. For a zooming user interface, which supports three-level zooming, a user can simply zoom the content or the entire desktop to focus on an embedded object in order to have enough working space for using the application. The windowing system can observe the available display size for the embedded object and automatically switch form the content view to an embedded application. This way, users can work with different applications within a single document and transparently switch between them in a hierarchical tree of embedded content objects. Figure 4 shows an example document in a zoomed out view (top left), a zoomed in view (top right), and the corresponding object hierarchy (bottom). The compound document embeds a bitmap graphic and an

illustration, whereas the illustration itself contains a bitmap graphic. The object hierarchy also shows the corresponding applications that are utilized to manipulate each of them.

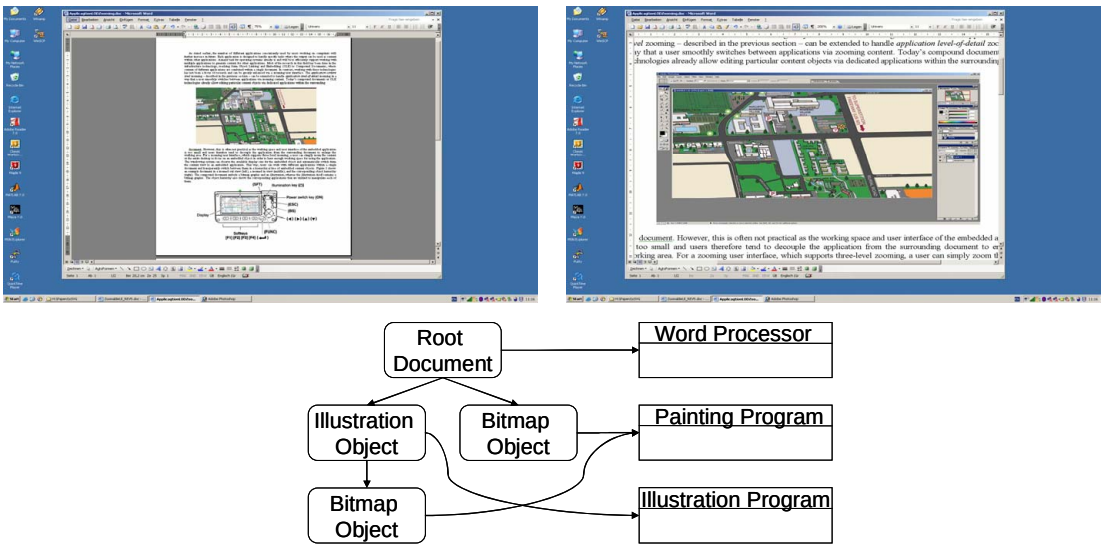

Figure 4: Overview of a compound document (top left), zoomed-in view for editing the embedded bitmap (top right), application/object hierarchy of a compound document (bottom)

The presented techniques help users to handle and use a great variety of advanced applications concurrently on a single desktop, especially if they are combined with adequate navigation techniques. Therefore, the following section presents important research in the field of efficient operation methods for zoomable user interfaces.

4. NAVIGATION METAPHORS IN ZOOMABLE INTERFACES

In this section we present two alterative concepts for navigation metaphors in zoomable user interfaces. The first concept is directly related to the *three-level zooming* that enables users to zoom the content, the application window, and the desktop independently from each other. The second concept is a *zoom bar* for zooming on the desktop level. Finally, we present an o*verview map* to keep the survey of all objects in the zooming world.

4.1 Operating with Three-Level Zooming

The navigation concept for three-level zooming is consistent on all zoom levels. The desktop level, application window level, and application content level can be zoomed and panned independently from each other. Therefore, each zoom level needs to have handles for zooming and panning. For the desktop we have added navigation areas at the top and the bottom (Figure 5). The areas are labeled with an explanation of their functionality.

Figure 5: Zoom areas at the bottom and the top of the desktop

To adjust the application window level and the application content level we have added navigation areas in the title and status bar to each window (Figure 6). This enables users to pan and zoom the window – by using the two left areas (a) – and separately its content – by using the right two areas (b). Within each group,

the left area controls the zooming and the right area is for panning. The title and the status bar are usually used to present textual information, therefore it is not possible to label the navigation areas with text. As a solution for this, different abstract presentations of their functionality are used for the four parts of the navigation areas.

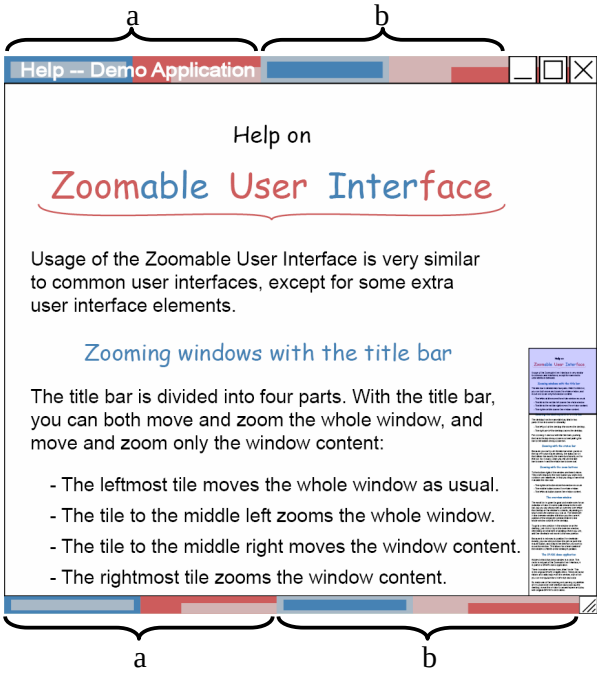

Figure 6: Zoom areas in the title and status bar of a window

The navigation in the zoomable user interface is handled by dragging. When the user clicks and drags the mouse over one part of the navigation area the position or zoom factor of the related level is adjusted. As feedback for the user the respective level is constantly updated while dragging the mouse. To achieve better response times anti aliasing could be disabled during the interaction process.

4.2 Zoom Bar

An alternative concept to the navigation used for *three-level zooming* is the *zoom bar* (Figure 7). This toolbar is rendered semitransparent in the foreground and positioned at the bottom of the desktop. The zoom bar appears if the user hits the bottom of the desktop with the mouse cursor or clicks on the small visible bar. The bar hides automatically if the mouse cursor leaves its focus area. To zoom in and out the user has to click on the respective button on the bar. Panning is combined with zooming and is controlled by clicking the button and moving the mouse cursor to the desired pan direction. As feedback for the user a line visualizes the current pan direction directly on the bar.

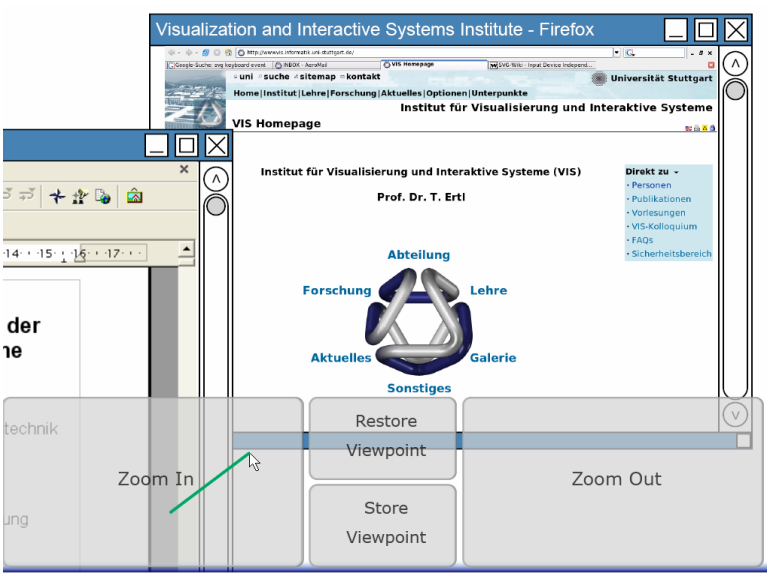

Figure 7: Zoom bar at the bottom of the desktop
(for evaluating the concept the content of the application windows are screenshots)

There are two further buttons on the zoom bar that can be used to store the current viewpoint, in order to restore it later. This feature enables users to store the current viewpoint before leaving it to look up some necessary information in another document somewhere on the desktop. Rapidly returning to the previous viewpoint is possible by only a single click on the *restore viewpoint* button.

4.3 Overview Map

Because the zoomable desktop can be arbitrarily larger than the area of the display and in the same way the application content can be arbitrarily larger than the application window, an overview map is presented in the lower right corner of each level (Figure 8). This is essential for users to survey the context of the presented information. The overview map for the desktop shows the entire virtual desktop at a smaller scale and the overview map for each application window shows the application's content at a smaller scale. A viewfinder in the overview map indicates the visible parts. The viewfinder can be dragged in the overview map to translate the viewpoint. Therefore, each position of the virtual desktop and the application content can be reached with one simple drag operation. The overview map covers just about 5% of the display size of application window size and is continuously updated. To save screen real estate the overview map hides itself if all information fits into the available desktop or application window.

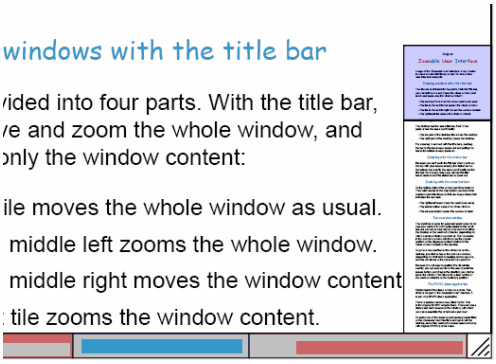

Figure 8: Overview map with a viewfinder in the lower right corner of a window

5. PROTOTYPE IMPLEMENTATION

Our prototypical implementation of a zoomable user interface is an extension to the SPARK framework. SPARK implements a widget set using SVG, including windows that are moveable and resizable and widgets, like buttons and sliders. SPARK follows to the Model View Controller (MVC) paradigm, which separates data, presentation, and control in the implementation. SVG supports a hierarchical modeling of graphical content, this paradigm is used by implementation of SPARK. Therefore, all windows are children of the desktop, and every content object of an application window is a child of its parent window. This property simplifies the implementation of the zoom and pan functions in zoomable user interfaces, because child objects inherit transformations of parent objects. As our prototype only changes the presentation aspect of SPARK, every application using the SPARK framework can transparently make use of the zoomable user interface prototype and benefit from its capabilities.

An example scenario of a zoomable desktop with two windows is given in the following. The position of each concept is defined by the “translate” and the scaling by the “scale” parameter.

```
<g id="desktop" transform="translate(d_x, d_y) scale(d_z)">
   [declaration of desktop elements]
  <g id="window1" transform="translate(w_x, w_y) scale(w_z)">
    [declaration of window1's elements]
   <g id="window1-content" transform="translate(c_x,c_y) scale(c_z)">
     [declaration of window1-content's elements]
   </g>
  </g>
  <g id="window2" [...] />
    [...]
</g>
```

The prototype attaches event handlers to all the user interface elements in order to initiate operations that have to be performed on different user events. The operations can be window manager internal operations like movements, translations, or scaling of the desktop, windows, or content of windows. The operations can also be external events that communicate user interaction to the application, e.g. a click on a button.

The unmodified SPARK framework changes the translate value of the window's transform attribute when the user drags the window's title bar and changes the window size when the user drags the resize handle at the lower right corner. Our prototype further enhances this approach and changes the translate and scale value of the window, the window's content, or the desktop. Furthermore, SPARK’s scrollbar of the window has been replaced by an overview map which is implemented by a translated and scaled reference to the window description:

```
<g id="window1-overview-content" transform="translate(o_x, o_y)
scale(o_z)" >
    <use xlink:href="#window1-body-content" />
</g>
```

Every time the window's content is changed, the translate and scale values for the overview map are recalculated in a way that 5% of the window's surface are covered. The overview map of the desktop shows all windows that are active on the desktop. Therefore, the bounding boxes of all windows have to be calculated, as they are used to determine the size and scaling in the overview map. The overview maps are recalculated on initial window creation, on events, and every time the window content changes.

6. CONCLUSION

We have shown how zoomable user interfaces can help users to work with multiple applications simultaneously. The concept of three-level zooming that we have proposed unifies the zooming functionality of different applications via the support of zooming within the underlying window manager. To control the zooming levels a novel technique is integrated which is also suitable for mobile devices – e.g. PDAs or Smartphones – with pen interfaces. In addition, we have presented a navigation method for zoomable user interfaces, independent of the three-level zooming paradigm. The zoom bar provides an enhanced visual feedback for zooming interaction, whereas the overview map follows the focus+context concept and helps the users to maintain an overview to the entire context.

To prove the proposed concepts we have implemented a prototype using SVG with the SPARK widget set. We were able to realize all proposed techniques using only SVG provided functionality. The prototype utilizes the key feature of SVG – a system independent vector graphics description – to deploy the zoomable user interface to different devices. As there are only prototypical widget sets available in SVG and, as our prototype is based on SPARK, there are still some commonly used widgets missing. The main problem for applications in SVG and even more for a zoomable user interface is the availability of efficient SVG viewers. There is still no hardware supported SVG viewer available, which is especially problematic for mobile devices, as they have only reduced computation power.

In future, we will focus on a practical implementation of the SVG-based zoomable user interface via the integration of existing applications. A remote desktop tool – e,g, VNC (Virtual Network Computing) – can be used to retrieve the image of a running application [VNC]. This image could be integrated into the SVG application window in the SVG-based user interface. Of course the events have to be send back to the existing application to achieve interactivity. This way, users will benefit form a zoomable user interface, even when using existing non-SVG-based applications.

REFERENCES

[Archy] Raskin Center for Humane Interfaces: *Archy*, http://rchi.raskincenter.org

[Bederson et al. 1994] Bederson, Benjamin B.; Hollan, James D.: Pad++ – a zooming graphical interface for exploring alternate interface physics, *Proceedings of the 7th annual ACM symposium on User interface software and technology*, pp. 17-26, 1994

[Bederson et al. 2000] Bederson, Benjamin B.; Meyer, Jon; Good, Lance: Jazz- an extensible zoomable user interface graphics toolkit in Java, *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pp. 171-180, 2000

[Bederson et al. 2004] Bederson, Benjamin B.; Grosjean, Jesse; Meyer, Jon: Toolkit Design for Interactive Structured Graphics, *IEEE Transactions on Software Engineering*, Volume 30, Issue 8, pp. 535-546, 2004

[Gutwin et al. 2004] Gutwin, Carl; Fedak, Chris: Interacting with big interfaces on small screens: a comparison of fisheye, zoom, and panning techniques, *Proceedings of the 2004 Conference on Graphics Interface*, pp. 145-152 2004

[Hightower et al. 1998] Hightower, Ron R.; Ring, Laura T.; Helfman, Jonathan I.; Bederson, Benjamin B.; Hollan, James D.: Graphical multiscale Web histories – A study of PadPrints, *Proceedings of the Ninth ACM Conference on Hypertext*, S. 58-65, 1998

[Hornbæk et al.] Hornbæk, Kasper; Bederson, Benjamin B.; Plaisant, Catherine: Navigation patterns and usability of zoomable user interfaces with and without an overview, *ACM Transactions on Computer-Human Interaction*, Volume 9, Issue 4, pp. 362-389, 2002

[Lewis et al. 2003] Lewis, Christopher T.; Mansfield, Philip A.; MacDonald, Glenn; Fettes, Alastair: SVG Programmers Application Resource Kit, *Proceedings of the SVG Open 2003 Conference, Vancouver*, Canada, 2003

[Perlin et al. 1993] Perlin, Ken; Fox, David: Pad – An Alternative Approach to the Computer Interface, *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, pp 57-64, 1993

[Raskin 2000] Raskin, Jef: *The Humane Interface – New Directions for Designing Interactive Systems*, Addison-Wesley, Boston, 2000

[SPARK] Lewis, Christopher T.; Fettes, Alastair; Mansfiel, Philip; Peto, *Chris: SVG Programmers' Application Resource Kit (SPARK)*, http://spark.sourceforge.net/

[SWiNE] Rosyada, Amri: *SWiNE – SVG Widget New Edition*, http://www.mycgiserver.com/~amri/swine/swine.htm

[Tactile 3D] Upper Bounds Interactive Inc.: *Tactile 3D*, http://www.tactile3d.com/

[VNC] RealVNC Ltd, *About RealVNC*, http://www.realvnc.com/