# The Hierarchical Visualisation System (HVS)

Keith Andrews, Werner Putz, and Alexander Nussbaumer
IICM, Graz University of Technology, Austria
kandrews@iicm.edu

## Abstract

*Numerous techniques have been developed for visualising hierarchically structured information. This paper presents a new framework for the visualisation of hierarchies called the Hierarchical Visualisation System (HVS). HVS is a general framework implemented in Java. It provides a synchronised, multiple view environment for visualising, exploring and managing large hierarchies.*

*HVS reads hierarchies either from the file system or from TreeML files. Eleven hierarchy browsers have so far been implemented within HVS, including: traditional tree views, the classic Walker tree layout, information pyramids, treemaps, a hyperbolic browser, sunburst, and cone trees.*

*In addition to being a tool to explore and manage hierarchies, HVS was also designed to provide a platform for the empirical evaluation and comparison of different hierarchy browsers. The Hierarchical Visualisation Testing Environment (HVTE) is a semi-automated testing environment built on top of HVS. HVTE is being used for a series of comparative studies of hierarchy browsers.*

**Keywords**: information visualisation, hierarchies, trees, framework, Java, coordinated views, empirical test environment.

## 1. Introduction

Information visualisation seeks to visualise abstract information structures and spaces to make them more easily accessible and manageable. Information is often organised into hierarchies of one kind or another. Various techniques have been developed to improve access to hierarchically structured information, from the humble tree view (such as the Macintosh File Finder or Windows Explorer) to realistic 3d botanical renderings of trees [8].

This paper presents a new framework called the Hierarchical Visualisation System (HVS). HVS implements a growing palette of hierarchy browsers and allows them to be used and evaluated both individually and in combination.

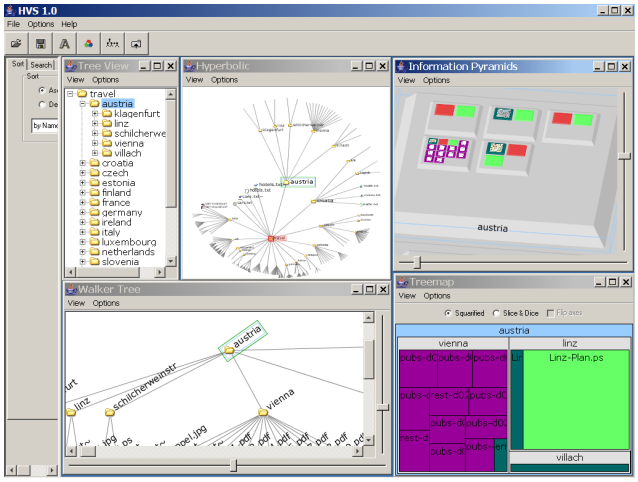


**Figure 1. The Hierarchical Visualisation System (HVS) provides multiple, synchronised views of a hierarchy. Five synchronised visualisations can be seen here: a classic Windows Explorer style tree view (top left), a hyperbolic browser (top middle), an information pyramids browser (top right), a Walker tree browser (bottom left), and a treemap browser (bottom right). The hierarchy is a tree of notes and documents concerning various travel destinations.**

## 2. Hierarchical Visualisation System (HVS)

HVS is a general framework implemented in Java, which provides a synchronised, multiple view environment for visualising, exploring and managing large hierarchies. Figure 1 shows an application of HVS, with five synchronised views (browsers) of a hierarchy. A more typical application might use two or three synchronised views.

Any browser can be specified as being Independent. It then no longer synchronises itself with the other browsers. Any further navigation or selection is local to that view. If, at a later point, that browser is switched back to

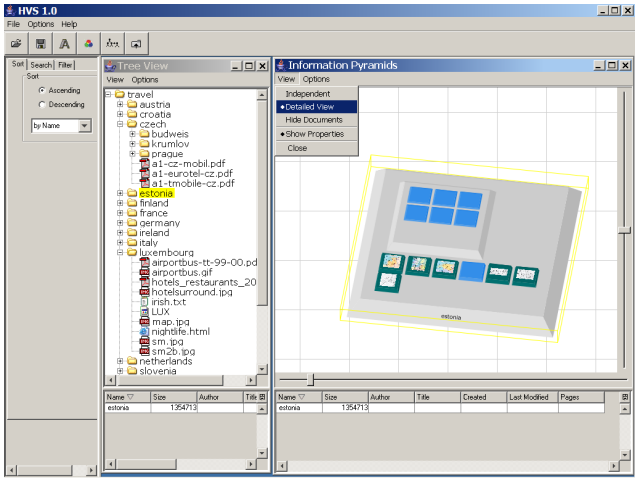


**Figure 2. HVS Detailed View. The pyramids browser on the right displays the details (contents) of the selected directory "estonia" in the tree view to the left.**

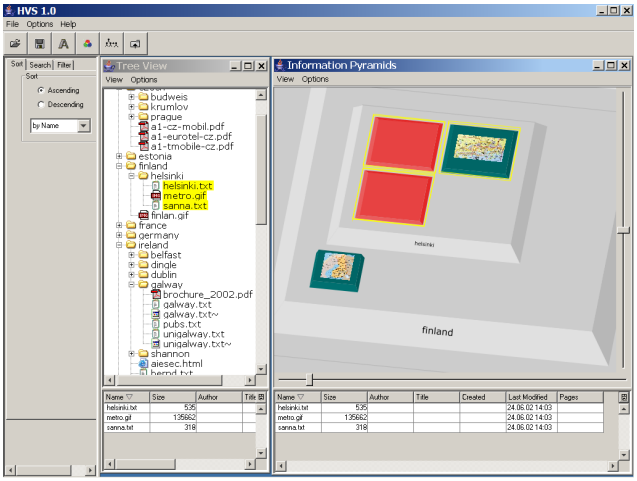


**Figure 3. HVS Selection. Three objects have been selected in the tree view on the left and are also selected in the synchronised pyramids browser to the right. The available metadata of the selected objects is shown in the properties table at the bottom of each view.**

Synchronised mode, it then re-synchronises itself with the rest of the browsers. Any browser can also be specified as being a Detailed View. If a single node is selected in any of the synchronised views, any browsers in Detailed View mode then display only the selected node and its children. This can be seen in Figure 2.

Selection and multiple selection (click, shift-click, and control-click) are possible in any view. Drag selection is also possible. Selections are propagated to all other synchronised views. Every browser supports the Show Properties function, which gives a tabular view of all of the available metadata for all selected objects. Multiple selection and Show Properties are illustrated in Figure 3.

HVS attempts to collect or extract as much metadata about objects in the hierarchy as possible. The metadata are extracted in a background thread and are cached in an XML format between HVS sessions. The various metadata fields are then available to all of the browsers within HVS. In a file system, each object's name, size, and last modification date are available. By looking at the object's extension, HVS attempts to deduce its type. The following metadata attributes are then generated for objects where possible: Author, Title, Subject, Number of Pages, Keywords, Comments, Creation Date, Document Modified, and File Modified. In particular, these fields may be available for PDF documents. In addition, thumbnail images in four different sizes are created from both images and from the first page of PDF documents.

The hierarchy is editable (rename, insert, delete) in principle, but it is configurable whether any edits are then saved to the original hierarchy. HVS also allows a set of external applications to be configured to to view documents of certain types. The external application is then started when the user activates Open Document (or middle-clicks).

The HVS framework provides centralised sorting, searching, and filtering functionality to all of its browsers. The panel to the left of the default HVS application provides access to this functionality to the user. Figure 4 shows a search for all objects with the extension ".gif". Eighteen such objects have been found, and the user is currently looking at number nine of eighteen. The set of matching objects is highlighted in blue, except for the currently focused search object, which is red. This is the image "inf-rostock-map.gif", which also happens to be selected (and thus is yellow).

## 3. The Architecture of HVS

The multiple view environment in HVS is based on the model-view-controller (MVC) design pattern [6]. As shown in Figure 5, the hierarchical data model (HDM) forms the centre of HVS. Input modules serve to construct the HDM from any one of a number of sources. In particular, reading from a file system and reading from a TreeML file are currently supported. Sort and filter functionality is implemented within the HDM.

The controller manages synchronisation of the various views. Each viewer must send events to the controller upon certain user actions, and must listen and respond to

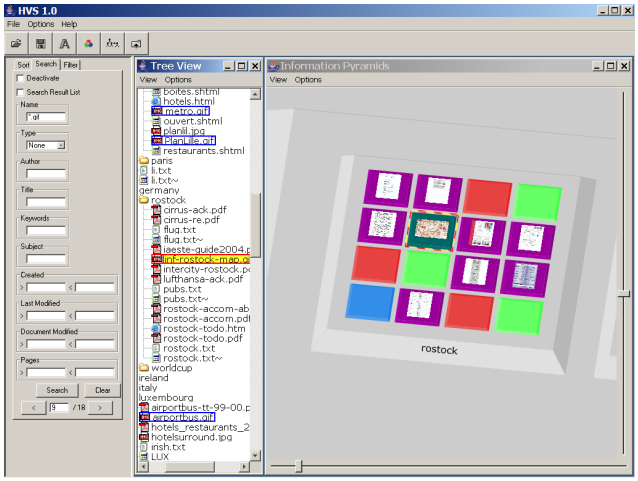


**Figure 4. Searching in HVS. The user is looking at object nine (red highlighting) of eighteen objects (blue highlighting) with the extension ".gif" in their name.**

events sent by the controller. The controller synchronises selection, expansion, and a change of focus (for example, through scrolling). Each browser is responsible for its own progressive rendering, gracefully degrading the display during navigation to maintain interactive frame rates.

HVS allows user configurations to be saved in an XML description file for loading at a later date. Hence, a particular hierarchy, a certain set of visualisations, synchronisation modes, colour codings, and the like can be preserved and reloaded in a single step.

## 4. Hierarchy Visualisations

Eleven different hierarchical visualisation techniques are currently implemented within HVS:

- *Classical Tree View*: The simplest visualisation within HVS is a classical tree view based on the Java JTree class. Directories and subdirectories can be expanded and collapsed and the entire view can be scrolled. See Figures 1 and 4.

- *Tree View+*: The Tree View+ is an extension of the standard tree view, with small circular icons indicating the relative size of each item. See Figure 7.

- *Walker Tree Browser*: The Walker tree browser is an implementation of the classic Walker tree layout algorithm [14]. The root of the tree is drawn at the top, its children are placed in a row beneath it, and so forth. See Figure 6.

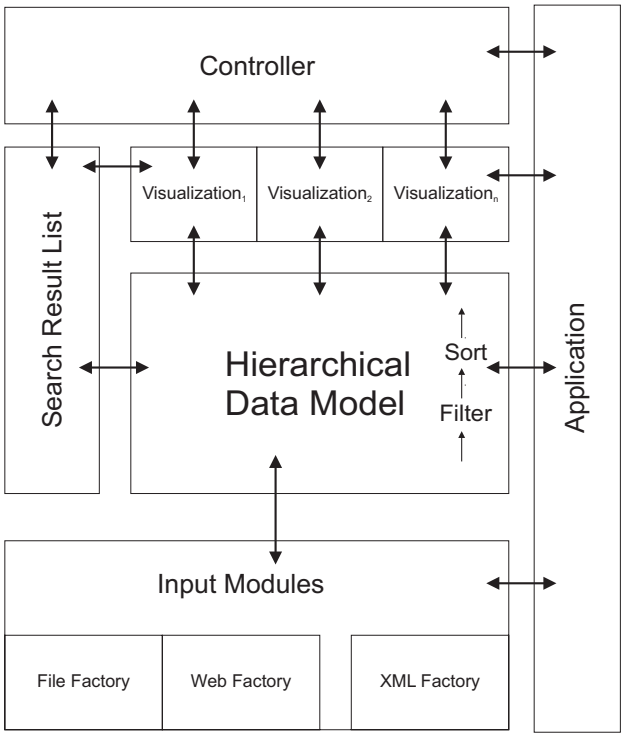


**Figure 5. The Architecture of HVS.**

- *Dendrogram Browser*: The Dendrogram Browser is similar to the Walker browser, but using horizontal and vertical edges, like in a dendrogram. See Figure 7.

- *Information Pyramids*: The information pyramids browser [1] is the most fully-featured browser in HVS. In information pyramids, the root directory forms the bottommost plateau. Its subdirectories are placed as plateaus upon it. Any documents (files) at each level are arranged as blocks at the front of the plateau. The thumbnails garnered by HVS are displayed on the upper surface of each document block.

  Free zooming with the mouse wheel is supported, allowing users to zoom to a particular part of the hierarchy, determined by the position of the mouse cursor. A change of focus causes the new focus object(s) to be brought to the centre of the display.

- *Treemap Browser*: Both squarified and slice and dice treemaps [12] are supported in the HVS treemap browser. Treemaps are laid out by recursively subdividing rectangles according to the size of their children. See Figure 8.

- *Hyperbolic Browser*: A home-grown version of the hyperbolic browser [10] was written from scratch. The

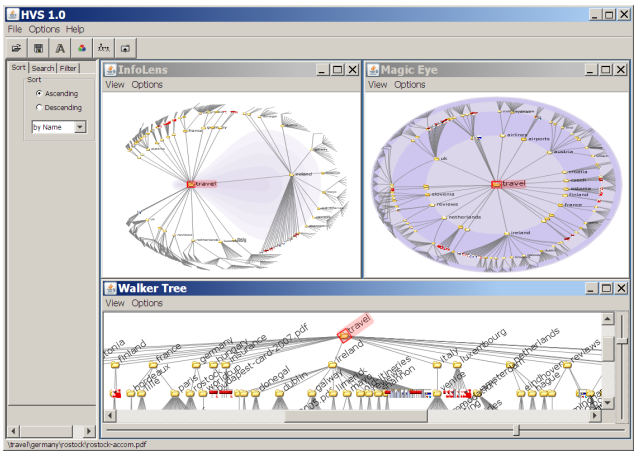


**Figure 6. HVS: The InfoLens browser (top left), magic eye browser (top right), and Walker tree browser (bottom).**

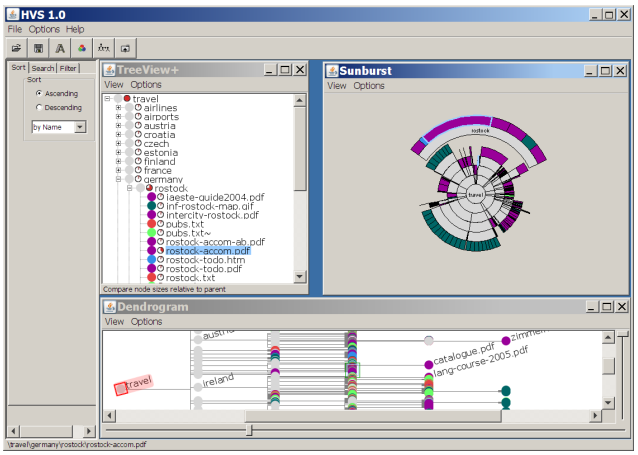


**Figure 7. HVS: The TreeView+ (top left), Sunburst browser (top right), and Dendrogram browser (bottom).**

tree is laid out in hyperbolic space and then mapped to the unit disc for display. See Figure 9.

- *Magic Eye View*: In the magic eye view [3, 9], a classic Walker layout is projected onto the surface of a hemisphere for display. See Figure 6.

- *InfoLens*: The HVS InfoLens is a new visualisation. A pear-shaped magnifying lens is fixed in the centre right of the display and a Walker tree layout can be dragged over it to magnify a certain area of interest. See Figure 6.

- *Sunburst*: The Sunburst browser is our own implementation of Sunburst [13]. See Figure 7.

- *Cone Tree*: A prototype implementation of the cone tree [11] technique has been written for HVS in Java and OpenGL. In the vertical cone tree layout, children are recursively placed around the base of a cone emanating from their parent. In the alternative horizontal cam tree, layout proceeds from left to right. The HVS cone tree browser can be seen in Figure 10.

## 5. Extending HVS

Extending HVS with a new hierarchy browser is fairly straightforward, owing to the plugin architecture of HVS. Assuming a new tree view visualisation is to be implemented for HVS, the first step is to create a new subdirectory in the plugin directory of HVS. In this newly created directory a plugin configuration file (XML) is created, as shown in Listing 1.

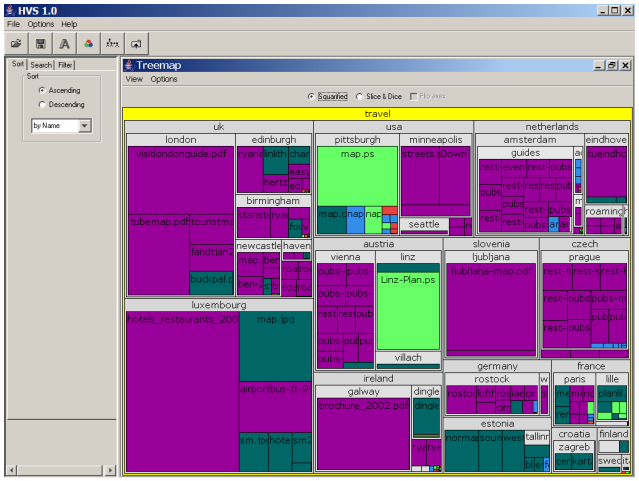


**Figure 8. The HVS Treemap Browser.**

The new TreeView class must then extend class Visualization and implement the ExpansionListener interface, as shown in Listing 2.

To achieve full synchronisation with other views, the new TreeView has both to listen to HVS events and to send events to HVS on the following user interactions:

- The user changes the current selection in the new tree view.

- A node is expanded or collapsed.

- The user scrolls in the tree.

To send these events, class Visualization provides the methods shown in Listing 3. After the TreeView has been com-

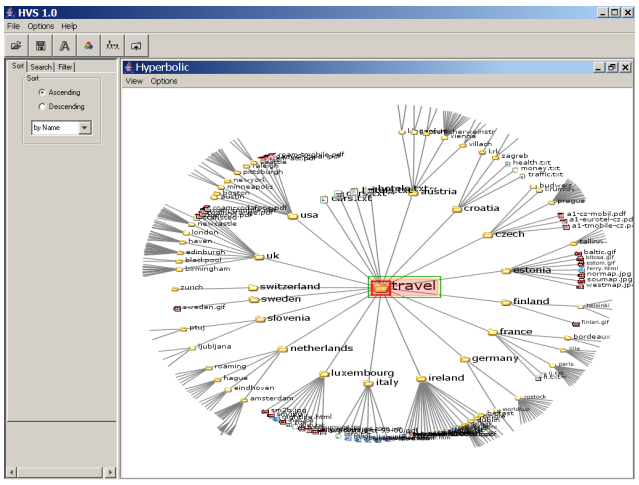


**Figure 9. The HVS Hyperbolic Browser.**

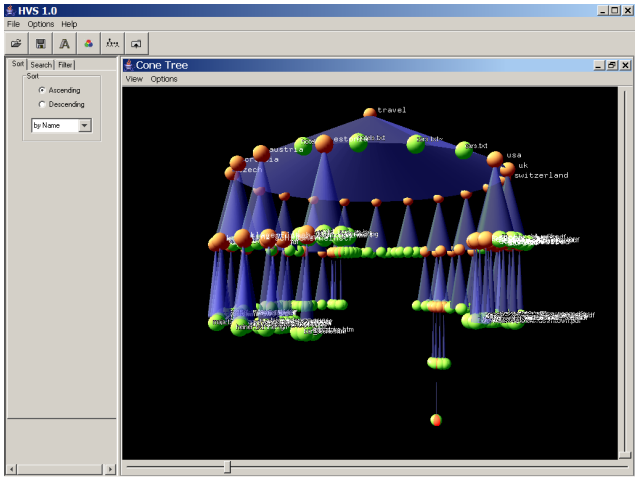


**Figure 10. The HVS cone tree browser.**

piled, its compiled class files are placed into the correct plugin subdirectory.

## 6. Hierarchical Visualisation Testing Environment (HVTE)

The Hierarchical Visualisation Testing Environment (HVTE) is a semi-automated testing environment built on top of HVS. HVTE uses the browser infrastructure of HVS and an SQL database to store and retrieve test cases. HVTE systematically presents one or more HVS browsers in sequence, each with a particular hierarchy and a particular user task, according to the test case randomly assigned to each user. HVTE then automatically logs answers entered by the user and task completion times. HVTE and the results of the first comparative study are described in the companion paper [2].

```
<?xml version="1.0" encoding ="UTF–8"?>
<plugin>
  <runtime>
    <path name="classes/"/>
    <extension point="iicm.hvs.
        visualization.Visualization"
      name="New Tree View"
      class="iicm.hvs.visualization.
          jtree.TreeView"
    />
  </runtime>
</plugin>
```

**Listing 1. HVS plugin configuration file.**

## 7. Related Work

The Information Visualization CyberInfrastructure (IVC) from Indiana University [4] is a fairly loose Java framework for information visualisation. IVC provides an XML-based interchange format, so that different visualisations can register and then interact with the same dataset. Treemaps, radial trees, and a hyperbolic browser have been implemented for hierarchies, but IVC does not focus specifically on hierarchical information.

The InfoVis Toolkit (IVTK) [5] is an excellent general Java toolkit for information visualisation. It uses an efficient table-based internal data representation. IVTK is more general than HVS, supporting graph and tabular structures. Treemaps and radial trees are available for hierarchical information, but IVTK is not optimised specifically for hierarchies.

prefuse [7] is an extremely flexible Java framework for information visualisation based on a visualisation pipeline. It uses an underlying graph model for its internal data representation. Developers build an interactive visualisation by chaining together components. Hierarchy browsers implemented in prefuse include radial, treemap, hyperbolic, and degree-of-interest trees. Again, prefuse is not focussed specifically on hierarchical information.

## 8. Concluding Remarks

This paper presented the Hierarchical Visualisation System (HVS), a framework for the visualisation of hierarchical structures. HVS is intended as a reference collection for hierarchical visualisation techniques, as a testbed for new hierarchical visualisation techniques, and as an environment for running comparative studies of hierarchy browsers.

```
public class TreeView extends
    Visualization
  implements ExpansionListener
{
  private JTree tree_;
  public TreeView(DataModel dataModel,
      Controller controller,
               SearchResult searchResult,
                    PopUpHandler
                  popUpHandler,
               JFrame frame)
  {
    super(dataModel,controller,
        searchResult,popUpHandler,frame);
  ....
    tree_ = new HvsJTree(treeModel)
    add(new JScrollPane(tree_);
  ....
  }
}
```

**Listing 2. Implementing a new TreeView browser in HVS.**

In the future, it is planned to implement further hierarchy browsers, including information slices, a radial node-link layout, a botanic visualisation, and a recursive voronoi browser. HVS will allow many different techniques to be tried out and evaluated, both individually and as combinations of synchronised browsers.

## References


[1] K. Andrews. Visual exploration of large hierarchies with information pyramids. In *Proc. Sixth International Conference on Information Visualisation (IV'02)*, pages 793–798, London, England, July 2002. IEEE Computer Society Press.

[2] K. Andrews and J. Kasanicka. A comparative study of four hierarchy browsers using the hierarchical visualisation testing environment (HVTE). In *Proc. 11th International Conference on Information Visualisation (IV'07)*, Zurich, Switzerland, July 2007. IEEE Computer Society Press.

[3] T. Burger. Magic eye view: Eine neue fokus + kontext technik zur darstellung von graphen. In german, University of Rostock, Institute of Computer Graphics, Apr. 1999.

[4] K. Börner and Y. Zhou. A software repository for education and research in information visualization. In *Proc. Fifth International Conference on Information Visualisation (IV'01)*, pages 257–262, London, England, July 2001. IEEE Computer Society Press.

[5] J.-D. Fekete. The infovis toolkit. In *Proc. IEEE Symposium on Information Visualization (InfoVis 2004)*, pages 167–174, Austin, Texas, USA, Oct. 2004.


```
protected final void fireFocusChanged(
    FocusEvent event)
protected final void
    fireSelectionChanged(SelectionEvent
    event)
protected final void
    fireCollectionExpanded(ExpansionEvent
     event)
protected final void
    fireCollectionCollapsed(
    ExpansionEvent event)
```

**Listing 3. Callbacks to signify user events.**


[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[7] J. Heer, S. K. Card, and J. A. Landay. prefuse: a toolkit for interactive information visualization. In *Proc. CHI 2005*, Portland, Oregon, USA, Apr. 2005. ACM.

[8] E. Kleiberg, H. van de Wetering, and J. van Wijk. Botanical visualization of huge hierarchies. In *Proc. IEEE Symposium on Information Visualization (InfoVis 2001)*, pages 87–94, San Diego, California, USA, Oct. 2001. IEEE Computer Society.

[9] M. Kreuseler, N. Lopez, and H. Schuhmann. A scalable framework for information visualization. In *Proc. IEEE Symposium on Information Visualization (InfoVis 2000)*, pages 27–36, Salt Lake City, Utah, USA, Oct. 2000. IEEE Computer Society.

[10] J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proc. CHI'95*, pages 401–408, Denver, Colorado, USA, May 1995. ACM.

[11] G. G. Robertson, J. D. Mackinlay, and S. K. Card. Cone trees: Animated 3D visualizations of hierarchical information. In *Proc. CHI'91*, pages 189–194, New Orleans, Louisiana, USA, May 1991. ACM.

[12] B. Shneiderman and M. Wattenberg. Ordered treemap layouts. In *Proc. IEEE Symposium on Information Visualization (InfoVis 2001)*, pages 73–78, San Diego, California, USA, Oct. 2001. IEEE Computer Society.

[13] J. T. Stasko and E. Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Proc. IEEE Symposium on Information Visualization (InfoVis 2000)*, pages 57–65, Salt Lake City, Utah, USA, Oct. 2000. IEEE Computer Society.

[14] J. Q. Walker, II. A node-positioning algorithm for general trees. *Software: Practice and Experience*, 20(7):685–705, July 1990.