ThreadVis for Thunderbird

A Thread Visualisation Extension
for the Mozilla Thunderbird Email Client

Alexander C. Hubmann-Haidvogel

ThreadVis for Thunderbird

A Thread Visualisation Extension
for the Mozilla Thunderbird Email Client

Master's Thesis

at

Graz University of Technology

submitted by

Alexander C. Hubmann-Haidvogel

Institute for Information Systems and Computer Media (IICM),
Graz University of Technology
A-8010 Graz, Austria

2nd September 2008

Advisor: Ao.Univ.-Prof. Dr. Keith Andrews, Graz University of Technology
Co-Advisor: Dr. Matthew Chalmers, University of Glasgow

ThreadVis für Thunderbird

Eine Thread-Visualisierungs-Erweiterung
für das Mozilla Thunderbird Email-Programm

Masterarbeit

an der

Technischen Universität Graz

vorgelegt von

Alexander C. Hubmann-Haidvogel

Institut für Informationssysteme und Computer Medien (IIICM),
Technische Universität Graz
A-8010 Graz

2. September 2008

Diese Arbeit ist in englischer Sprache verfasst.

Begutachter: Ao.Univ.-Prof. Dr. Keith Andrews, TU Graz
Mitbetreuer: Dr. Matthew Chalmers, University of Glasgow

Abstract

Good ideas from the field of information visualisation sometimes fail to make it into everyday interfaces used by real people. ThreadVis incorporates the idea of a Thread Arcs visualisation of email conversations (or threads) into the popular Mozilla Thunderbird email client.

Implemented as a Thunderbird add-on in JavaScript and XUL, the ThreadVis extension displays all messages in an email or newsgroup discussion in a compact visual map. The chronology of and contributors to a discussion can be easily identified. The visualisation supports user interaction in navigating between messages in a thread and is synchronised with the main Thunderbird application. It is also possible to export the visual thread map in scalable vector graphics (SVG) format.

Two forms of usability evaluation were used during ThreadVis development. A usage study gathered log data from 25 test participants and analysed user behaviour patterns with the software. A thinking aloud test with six test users was run to provide formative feedback.

Kurzfassung

Gute Ideen aus dem Feld der Informations-Visualisierung scheitern manchmal daran, in alltägliche Benutzeroberflächen integriert zu werden, die von echten Personen verwendet werden. ThreadVis bindet die Idee einer Thread Arcs Visualisierung von Email-Konversationen (oder Threads) in das populäre Email-Programm Mozilla Thunderbird ein.

Als Thunderbird Add-On in JavaScript und XUL implementiert, stellt die ThreadVis Erweiterung alle Nachrichten einer Email- oder Newsgruppen-Diskussion in einer kompakten Karte dar. Die Chronologie und die Beteiligten einer Diskussion können leicht identifiziert werden. Die Visualisierung unterstützt Benutzer-Interaktion für das Navigieren zwischen Nachrichten eines Threads und ist mit der Thunderbird-Applikation synchronisiert. Es ist ebenso möglich, die Thread-Abbildung im Scalable Vector Graphics (SVG) Format zu exportieren.

Zwei Arten von Usability-Evaluierung wurden während der Entwicklung von ThreadVis verwendet. Eine Benutzer-Studie sammelte Log-Daten von 25 Test-Personen und analysierte Muster im Benutzerverhalten mit der Software. Ein Thinking-Aloud Test mit sechs Test-Personen wurde durchgeführt, um formative Rückmeldungen zu bekommen.

Pledge of Integrity

I hereby certify that the work presented in this thesis is my own, that all work performed by others is appropriately declared and cited, and that no sources other than those listed were used.

Place: _____

Date: _____

Signature: _____

Ich versichere ehrenwörtlich, dass ich diese Arbeit selbständig verfasst habe, dass sämtliche Arbeiten von Anderen entsprechend gekennzeichnet und mit Quellenangaben versehen sind, und dass ich keine anderen als die angegebenen Quellen benutzt habe.

Ort: _____

Datum: _____

Unterschrift: _____

Contents

List of Figures

List of Tables

Listings

Acknowledgements

First and foremost I have to thank my wife Iris for supporting me throughout this thesis. Without her encouragement, this thesis would not have been possible.

I want to thank Keith Andrews, who gave me the chance to write part of this thesis in Glasgow and Matthew Chalmers, who supported me at the University of Glasgow.

I also have to thank the numerous alpha- and beta-testers of the ThreadVis extension for being my guinea pigs, for reporting errors, crashes and their valuable feedback.

Alexander C. Hubmann-Haidvogel
Vienna, Austria, September 2008

Credits

I would like to thank the following individuals and organisations for permission to use their material:

- The layout of this thesis is based on Keith Andrew's skeleton thesis [Andrews, 2006].
- Chapter 5 is based on Kerr [2003a].

The following are used under the terms of the ACM Copyright Notice (see page xv):

- Figure 2.1 was extracted from Mackinlay et al. [1991].
- Figure 2.12 was extracted from Boardman [2000].
- Figure 2.14 was extracted from Robertson et al. [1991].
- Figure 2.21 was extracted from Rao and Card [1994].
- Figure 2.22 was extracted from Tweedie et al. [1994].
- Figure 4.3 was extracted from Li et al. [2004].
- Figure 4.4a was extracted from Li et al. [2004].
- Figure 4.4b was extracted from Li et al. [2004].
- Figure 4.7 was extracted from Viégas et al. [2006].
- Figure 4.8 was extracted from Viégas et al. [2006].
- Figure 4.9 was extracted from Venolia and Neustaedter [2003].

ACM Copyright Notice

Chapter 1

Introduction

This thesis presents the ThreadVis extension for Mozilla Thunderbird, a visualisation for email conversations.

Chapter 2 provides an overview over various information visualisation techniques. Different examples for visualising linear data, hierarchies, networks, multidimensional data as well as vector spaces are presented to illustrate the different techniques. A selection of the most common email readers are discussed in Chapter 3. The list contains both traditional, desktop mail clients, and pure web mail interfaces, which have spread widely in the last few years. Chapter 4 describes visualisations targeted specifically at email clients. These include both external applications, running parallel to the standard mail client, and visualisations which can be included in the mail client user interface itself. The Thread Arcs idea is discussed in great detail in Chapter 5, as it forms the base of the ThreadVis idea described later on. It is a compact visualisation, suited ideally for integration into a mail client user interface.

Chapter 6 introduces all the basics needed to develop a Mozilla Thunderbird extension. The user interface description language is presented as well as available APIs, localisation, package management, and managing automated updates. The Mozilla Thunderbird extension ThreadVis (based on Thread Arcs) is discussed in Chapter 7. At first, an overview of the features of the extension is presented. Implementation issues, obstacles, and specific visualisation ideas conclude the chapter. The implementation details of the ThreadVis extension are explained in Chapter 8. The description focuses on key elements of the implementation such as the calculation of email conversations, the visualisation itself, and the use of extensive caches to accelerate the extension.

Chapter 9 presents the results of an anonymous usage study. The ThreadVis extension includes a logging feature which records anonymous data about all user interactions and users were asked to send in the generated files regularly. The results of a thinking aloud test are discussed in Chapter 10. Six test users were asked to perform routine tasks in the Mozilla Thunderbird mail client. Their subjective impressions and their feedback is discussed as well as possible enhancements to the extension resulting from this feedback. Chapter 11 concludes with a list of open issues and possible future enhancements to the ThreadVis extension.

Chapter 2

Information Visualisation

"Overview first, zoom and filter, then details-on-demand."

[The Visual Information Seeking Mantra, Shneiderman [1996]]

In today's society, everyone is confronted with an ever increasing amount of data to process and understand. One way to manage this flood of information is information visualisation. The human visual perception system is capable of receiving large amounts of information and can automatically detect patterns and changes. This behaviour is utilised in information visualisation to offload processing of information on to the visual system. Hearst [2003] says that information visualisation is:

> "The depiction of information using spatial or graphical representations, to facilitate comparison, pattern recognition, change detection, and other cognitive skills by making use of the visual system."

Information visualisation tries to visualise abstract information that has no direct physical representation. This makes designing good visualisations even more difficult, as the same data can be represented in multiple ways using different mappings from an abstract space to a physical two- or three-dimensional space. According to Shneiderman [1996] and Andrews [2002b], information can be divided into the following categories:

- *Linear:* Program source code, alphabetical lists, chronological records, etc.
- *Spatial:* Two-dimensional data such as floor plans, geographic maps, etc; three-dimensional data such as CAD models, CAT scans, etc.
- *Hierarchies:* Tree structures such as family trees, file systems, and library catalogues.
- *Networks:* General graph structures, such as semantic networks, webs, social networks, etc.
- *Multidimensional Metadata:* Items with n attributes become points in a n-dimensional space. Examples are metadata attributes, data from relational or statistical databases, etc.
- *Feature-Based Vector Spaces:* A vector is constructed from the content of an item and is used to represent its characteristics. Using a similarity metric, objects can be placed in a thematic map, where similar objects are grouped by proximity.

Regardless of the type of information, many visualisation applications follow the Visual Information Seeking Mantra formulated by Shneiderman [1996]:

"Overview first, zoom and filter, then details-on-demand."

Visualisations can not only be categorised by the information they display, but also by the representation method used to display data. Focus-plus-context provides both local focus and surrounding context to help the user maintain orientation. This effect can be obtained either inherently by using a three-dimensional view where foreground objects are in focus and objects in the background provide context or explicitly by applying a geometric distortion, for example in a fisheye view [Furnas, 1986]. In overview-plus-detail visualisations, context is provided by a separate overview display. Space-filling techniques [Jerding and Stasko, 1998] use every pixel and even sub-pixels of the whole screen to represent information.

The rest of this chapter gives an overview of different information visualisation techniques, structured by the categories described above. Visualisations of spatial data are not discussed, as two- or three-dimensional data generally have a fairly obvious representation in 2d or 3d space. Section 2.1 discusses visualisation techniques for linear structures, Section 2.2 deals with visualisation techniques for hierarchical information, Section 2.3 describes network visualisations, Section 2.4 deals with multidimensional visualisation techniques and Section 2.5 discusses visualisation techniques for feature-based vector spaces.

2.1 Linear

2.1.1 Perspective Wall

The Perspective Wall visualisation [Mackinlay et al., 1991] was developed at Xerox PARC and further developed by Inxight Software [Inxight, 2006] (which is now part of BusinessObjects, an SAP company), renamed Time Wall. This visualisation presents linear information in a three-dimensional view. A two-dimensional layout is displayed on a three-dimensional wall, with the x-axis representing time and the y-axis representing other attributes of the visualised documents.

The central panel shows details about the focused section, the left and right panels provide context with the left panel showing information preceding the main section and the right panel showing information succeeding the main section. 3d perspective is used to achieve a focus-plus-context effect. The focus area is in the foreground and the context in the background. Figure 2.1 shows the initial Perspective Wall visualisation, Figure 2.2 shows the newer Time Wall visualisation. As Inxight has been acquired by Business Objects (an SAP company), the presented Java applet is no longer available for download.

2.1.2 SeeSoft

Seesoft [Eick et al., 1992] is an overview-plus-detail technique for visualising large amounts of source code. The visualisation provides different views on the code. The standard text view displays the source code. In the line view, every line of code is represented by a single row of pixels. The pixel representation compresses the line view in that every line of code is displayed not by a row of pixels but by a single or small number of pixels, making it possible to show over a million lines of code on a standard monitor [Ball and Eick, 1996]. The file summary representation presents file-level statistics, where each file is represented in a rectangle. The height of the rectangle is determined by the file size (number of lines). Finally, the hierarchical representation visualises the source code tree.

In each visualisation, different attributes can be colour coded, such as code age, author, date of creation, etc. Figure 2.3 gives examples of the text, line and pixel representations.

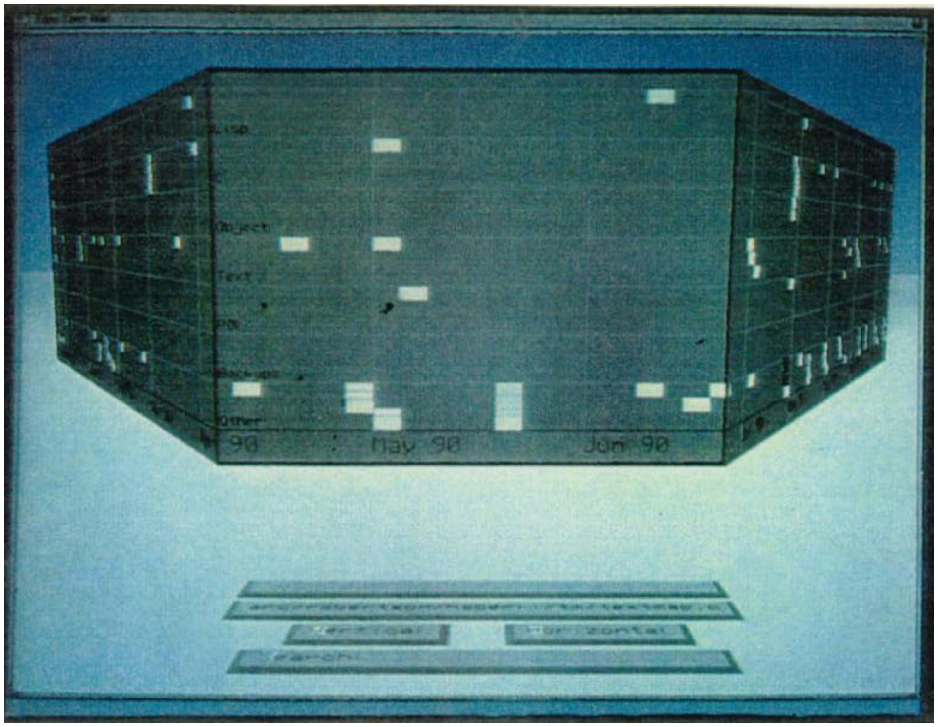


Figure 2.1: The Perspective Wall showing linear information in a three-dimensional view. The central panel shows details about the focused section, the panes to the left and right provide context. [Extracted from Mackinlay et al. [1991] under the terms of the ACM copyright. Copyright © by the Association for Computing Machinery, Inc.]

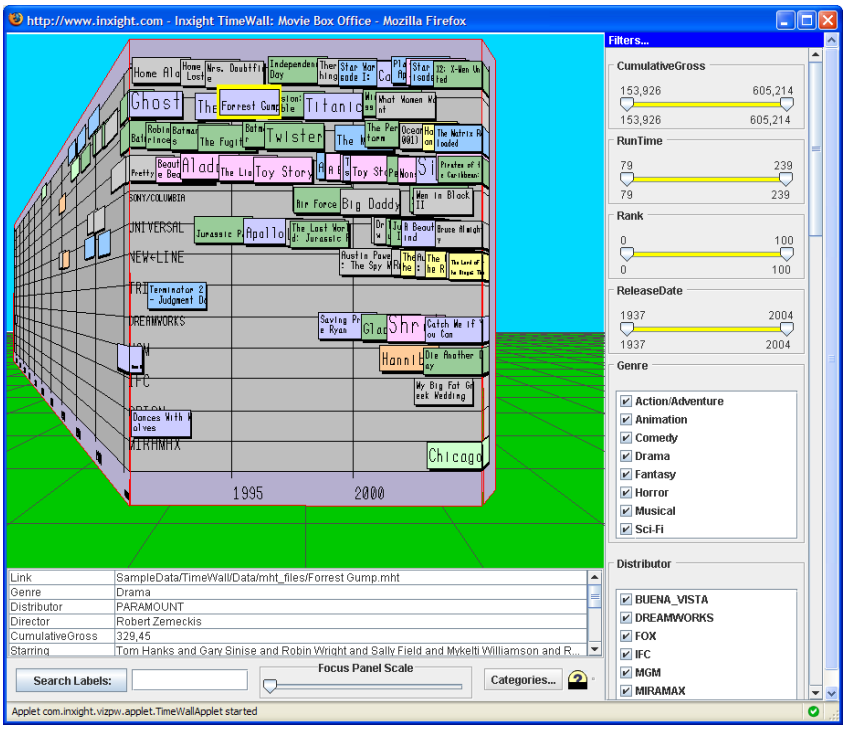


Figure 2.2: The Time Wall displaying movies released between 1937 and 2004. The main panel focuses on movies between 1990 and 2004. Details about the selected movie are displayed below the visualisation. Different genres are colour coded, sliders allow the selection of certain attributes. [Screen shot of demo applet at Inxight [2006]. Inxight is now part of Business Objects, the Java applet is no longer available.]

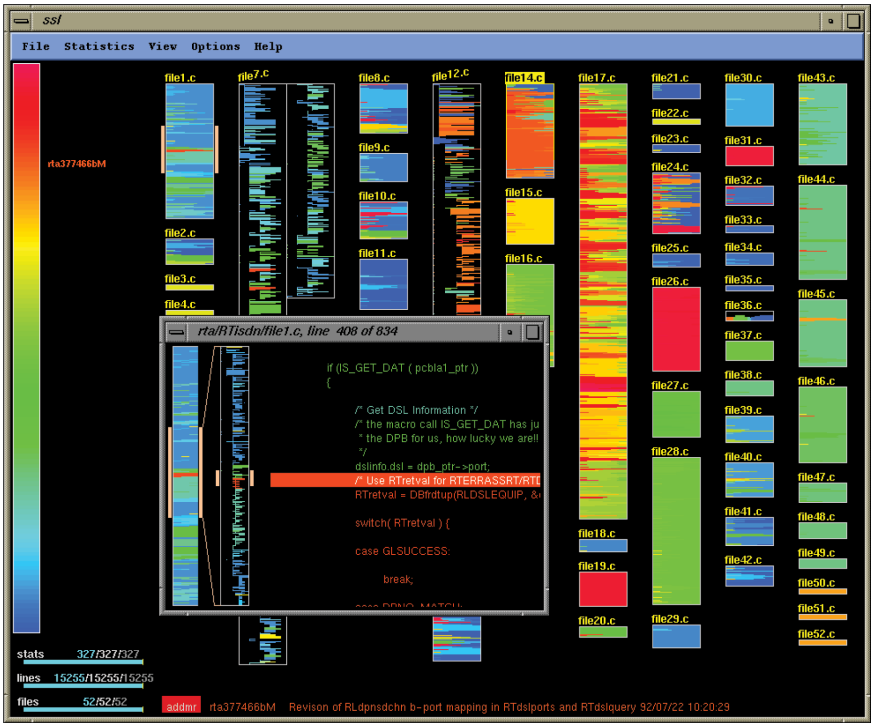


Figure 2.3: SeeSoft visualising 52 files with a total of 15,255 lines of codes. The newest lines are drawn in red, the oldest in blue. The browser window (smaller window) shows three different representations of code (from left to right): pixel, line and text representations. [Extracted from Ball and Eick [1996]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

2.1.3 Lifestreams

Lifestreams [Freeman and Fertig, 1995; Fertig et al., 1996] represent a diary of someone's electronic life. Every document ever created or received is stored in the lifestream, sorted chronologically. The tail of the stream contains documents from the past, moving towards the front, the stream contains more recent documents. Moving from the present into the future, the stream contains documents the user will need such as reminders, schedules or to-do lists.

A lifestream supports five different operations: `new`, `clone`, `transfer`, `find` and `summary`. `new` and `clone` create new documents. `transfer` copies documents to someone else's lifestream. `find` prompts for a search query and creates a substream, which is a view on a collection of documents which are relevant to the specified search query. `summary` takes a substream and compresses it into an overview document. Figure 2.4 shows a lifestream with the selected document displayed in detail.

2.1.4 Arc Diagrams

Arc Diagrams [Wattenberg, 2002] are able to visualise complex patterns of repetition in string data. An algorithm finds repeated substrings which are then displayed as translucent arcs. To avoid overly detailed visualisations, only a subset of all repetitions is shown. The corresponding intervals of each repetition are connected using a thick semi-circular arc. Figure 2.5 shows a simple example of an arc diagram visualising the repetition of different substrings.

The visualisation technique cannot only be applied to string data, but also to music, HTML pages, DNA sequences and even Java byte code. Figure 2.6 shows repetitions on the piece "Für Elise" by Ludwig van Beethoven visualised using arc diagrams. In music, a simplified version of the algorithm is used. Matches are based on similar pitch and in chords all but the top note is disregarded. Nevertheless

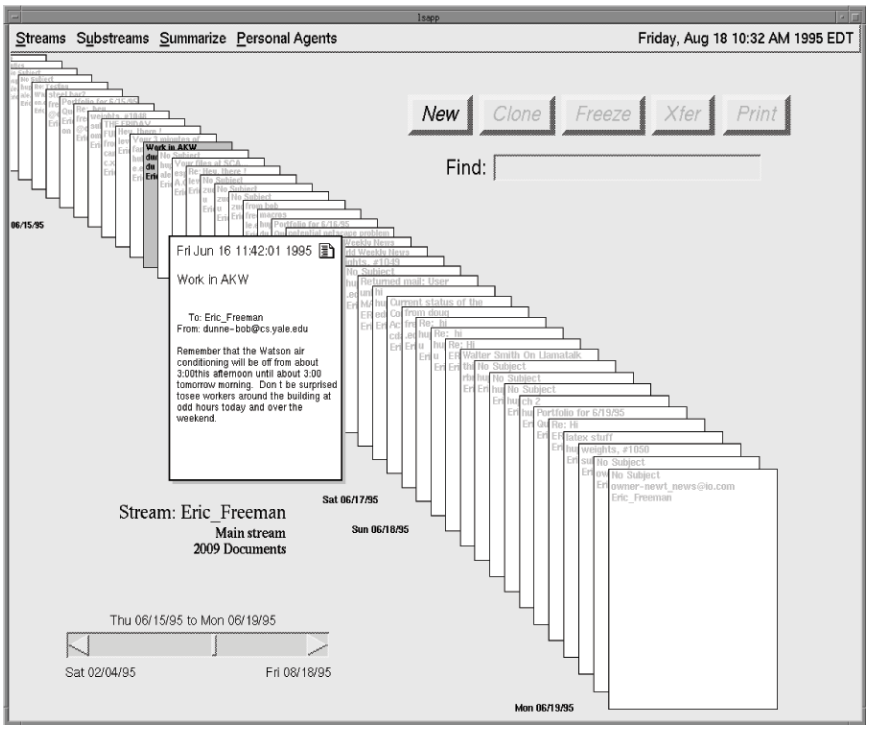


Figure 2.4: The Lifestreams interface, showing a stream of documents, sorted chronologically. The selected document is displayed in detail. [Extracted from Fertig et al. [1996]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

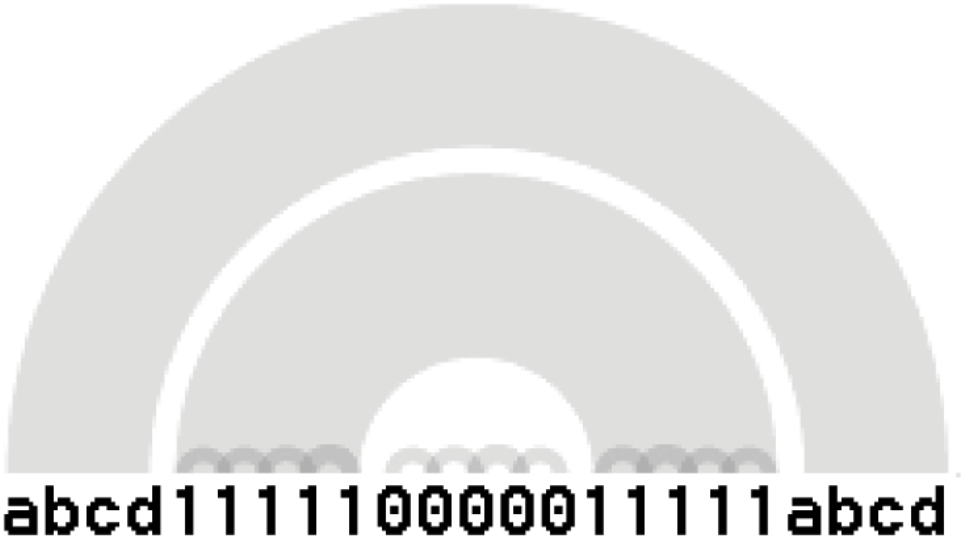


Figure 2.5: Arc Diagrams showing the repetition of different substrings. Repetitions are connected using thick semi-circular arcs. [Extracted from Wattenberg [2002]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

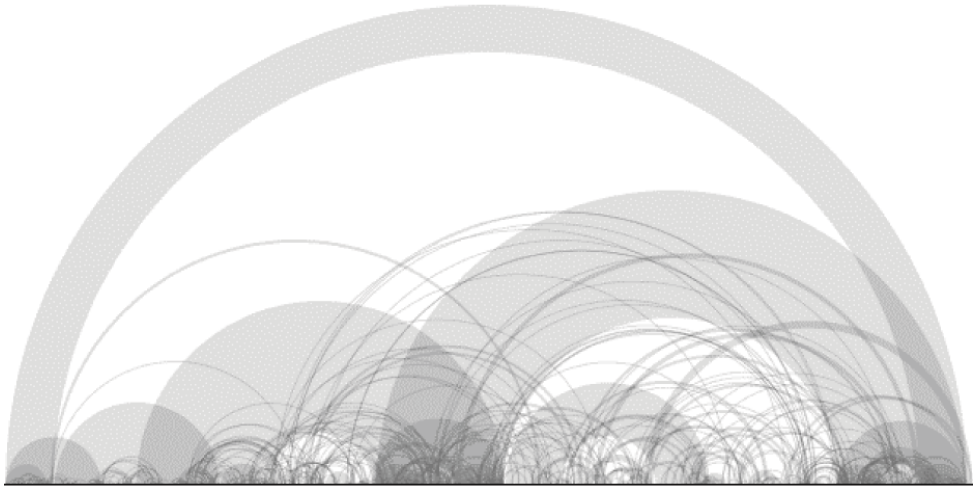


Figure 2.6: Arc Diagrams showing the piece "Für Elise". It can be clearly seen that the piece begins and ends with the same passage, with a longer version of the passage repeated throughout. [Extracted from Wattenberg [2002]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

the overall structure of the piece is revealed in the visualisation.

2.2 Hierarchies

2.2.1 Tree Views

Tree Views are simple visualisations for hierarchical data. A tree view on the left shows the structure of the hierarchy, a list view on the right shows the items at a particular level. File browsers are typically tree views, the most common example being the Windows Explorer in Microsoft Windows.

The hierarchy is visualised using a list of directories and files. Different levels of the hierarchy are represented by different amounts of indentation, with the child elements displayed beneath the parent element. Figure 2.7 shows a screen shot of Windows Explorer displaying a file hierarchy on the harddisk.

2.2.2 WebTOC

WebTOC [Nation et al., 1997; Nation, 1998] generates a tree view of the structure of a web site. It enhances the traditional tree view by adding supplementary statistical information. Coloured bars display the proportion of different media types, shadows beneath the bars represent the number of files at that level. The length of the bar is proportional to the size of the element. A single bar represents a file, a directory is represented by a group of bars. To accommodate small and large elements, a logarithmic scale is used to show the size of the elements in bytes. Figure 2.8 shows a web site on the right and the WebTOC visualisation of the web site's hierarchy on the left.

2.2.3 Hyperbolic Browsers

Hyperbolic Browsers [Lamping and Rao, 1994; Lamping et al., 1995] are a space-filling and focus-plus-context visualisation technique which supports displaying several levels of a hierarchy at the same time. The hierarchy tree is laid out on a hyperbolic plane and is mapped to the unit disc for display.

This makes it possible both to see the entire hierarchy for context, and to focus on a selected branch of the hierarchy to display details. The parts of the hierarchy at the edges are distorted and become

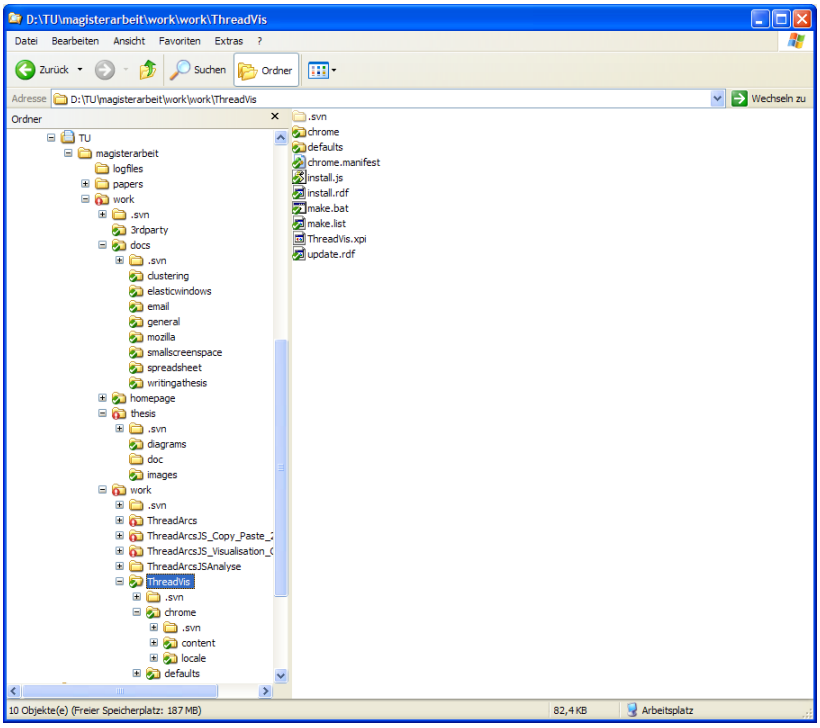


Figure 2.7: Windows Explorer displaying a section of a file system tree. Different levels of the hierarchy are displayed using different amounts of indentation. The contents of the selected folder are displayed in detail.

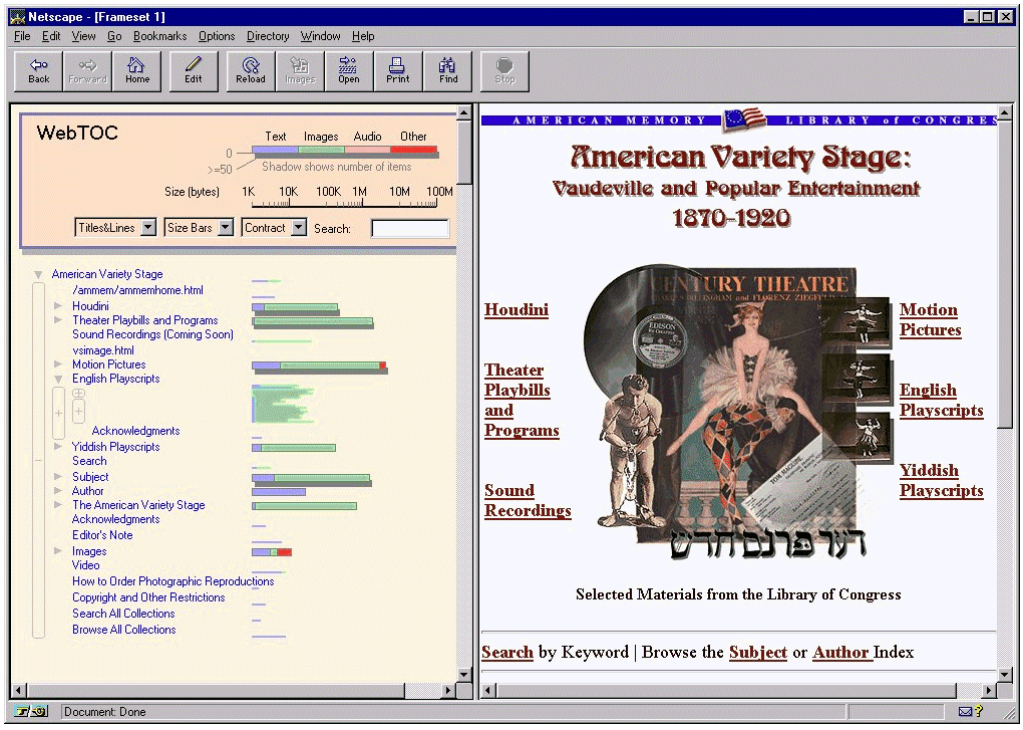


Figure 2.8: WebTOC displaying the structure of a web site on the left, the content is displayed on the right. Coloured bars represent different media types, the length of the bar is proportional to the size of the element. [Extracted from Nation et al. [1997]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

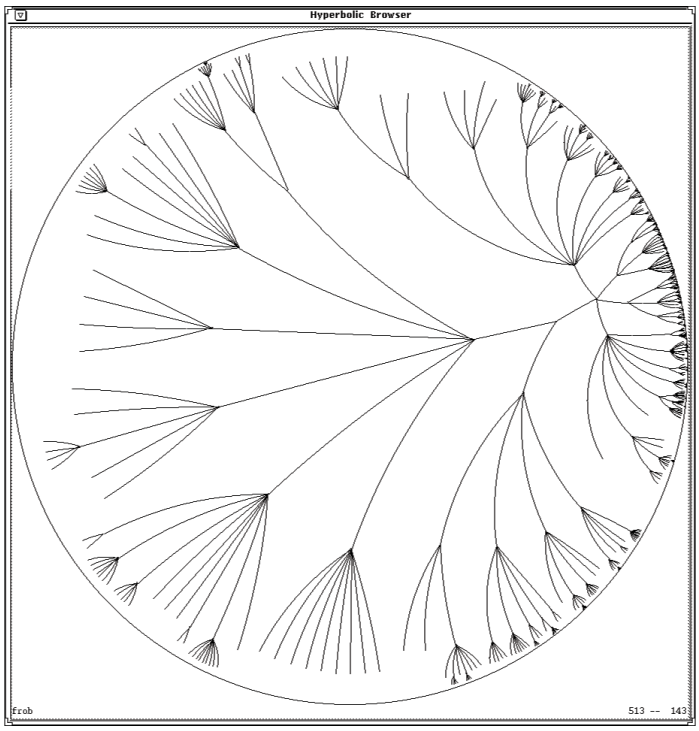


Figure 2.9: The Hyperbolic Browser displaying a tree with 1004 nodes. Focus is put on a selected branch, the rest of the hierarchy is distorted. [Extracted from Lamping et al. [1995]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

almost invisible, while the distortion area in the middle of the visualisation acts like a magnifying glass to display details. If a node is selected it is brought to the centre in an animated view to display details. Figure 2.9 shows a tree with 1004 nodes displayed using a Hyperbolic Browser.

2.2.4 Information Slices

The idea behind Information Slices [Andrews and Heidegger, 1998] is to lay out a hierarchy on a series of semi-circular discs. Each disc can represent one or more levels of the hierarchy, where the number of levels can be changed interactively. Deeper hierarchies which cannot be displayed using a single disc are represented using a cascading series of discs.

The area of each segment is proportional to the total size of its contents. When navigating through the hierarchy, the contents of a segment in the left disc is displayed on the right disc. In very deep hierarchies, the left disc can be miniaturised and slid off to the left and a new disc displaying the content is opened to the right. Figure 2.10 shows a directory in the Java JDK 1.1.6 distribution for Solaris. The `text/html` directory is selected in the left disc and its content is displayed in the right disc.

2.2.5 Sunburst

Sunburst [Stasko et al., 2000; Stasko and Zhang, 2000] developed the idea of Information Slices (see previous section) further. The visualisation uses a full disc to display the hierarchy. The different levels of the hierarchy are represented by concentric circles with the root node at the centre. Elements inside a directory are drawn inside the arc of the parent directory. Details can be displayed using different techniques:

- *Angular Detail*: When an item is selected, the whole hierarchy is shrunk and moved to the corner of the screen. This way space is provided for the selected item which is extended out of the hierarchy

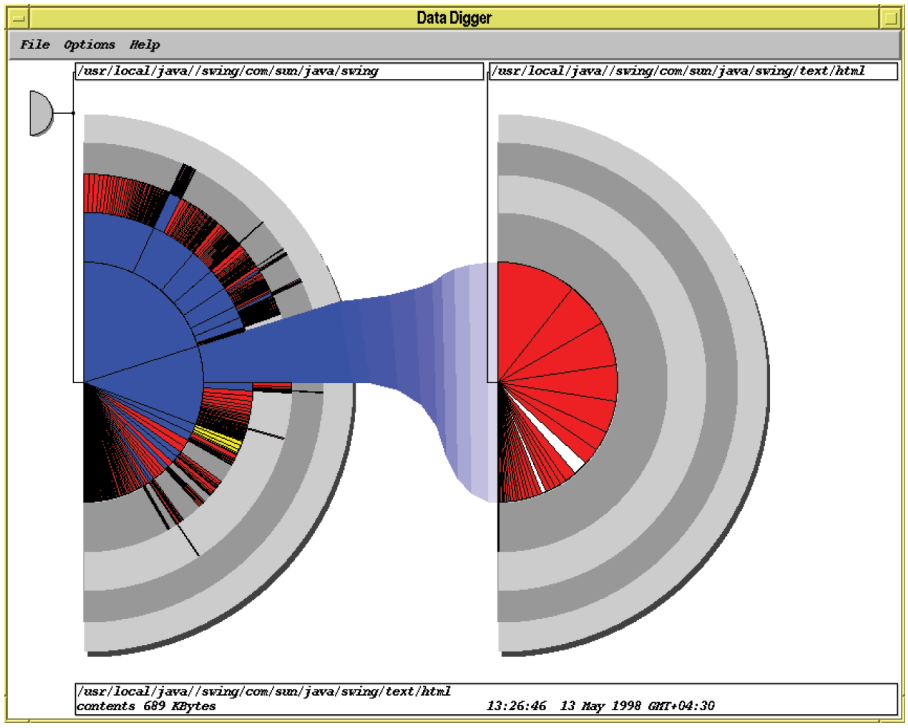


Figure 2.10: Information Slices displaying Sun Microsystem's JDK 1.1.6 for Solaris with 6158 files, 502 directories and a maximum depth of 9. The area of each segment is proportional to the size of its contents. [Extracted from Andrews and Heidegger [1998]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

and is represented by a larger arc.

- *Details Outside*: When an item is selected, the hierarchy is shrunk, but stays in the centre of the screen. The selected item is expanded and displayed in a 360 degree ring around the hierarchy, with sub-items represented by arcs.

- *Details Inside*: When an item is selected, the selection appears in the centre of the view and the hierarchy is pushed outwards. The selected item and its children are drawn from the centre outwards.

Elements in the visualisation can be colour-coded to indicate different file types, modification dates, or other meta-data attributes. Figure 2.11 shows a file system visualised using the Sunburst visualisation.

2.2.6 Bubble Tree

Bubble Trees [Boardman, 2000] provide a powerful focus-plus-context tree navigation mechanism in which trees recursively sub-categorise themselves into sub-trees. Each sub-tree is represented by a bubble, which encloses the root node and its descendants. Details can be obtained by one of three detail-increasing interactions:

- *Bursting* an opaque bubble reveals the root node and its sub-trees. See Figure 2.12, steps 1 to 2 and 2 to 3.

- *Probing* expands the selected bubble and contracts the others. Thus extra screen space is dedicated to the focus of interest. See Figure 2.12, step 3 to 4.

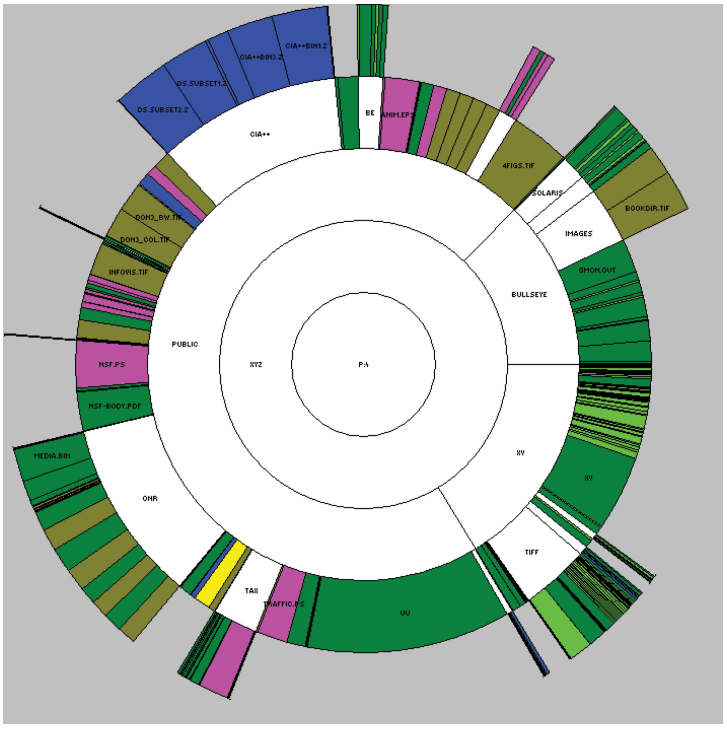


Figure 2.11: A Sunburst visualisation of a file system. Different levels of the hierarchy are represented by concentric circles, with the root node in the centre. Colour coding shows different file types or other meta-data attributes. [Extracted from Stasko and Zhang [2000]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

- *Focus* expands the selected bubble further and the other bubbles are abstracted into a hyperbubble. See Figure 2.12, step 4 to 5.

A backtracking mechanism is provided by the chain of hyperbubbles which are formed as the user navigates through the tree. Animating the interactions helps the user keep track.

2.2.7 Tree Maps

Tree Maps [Johnson and Shneiderman, 1991; Shneiderman, 1992] are space-filling visualisations which display hierarchies by alternately dividing the available screen space horizontally and vertically (also referred to as "slice-and-dice"). The area of each rectangle is proportional to the number or size of items it contains.

Using nested rectangles by recursively subdividing rectangles, deep hierarchical structures can be visualised. The outermost rectangle represents the root element or the highest level, sub-nodes are represented by rectangles included in other rectangles.

2.2.8 Market Map

The Market Map [Wattenberg, 1999; Schneiderman and Wattenberg, 2001] is a version of a tree map (see previous section). A heuristic is used to place the sub-rectangles to avoid narrow rectangles, which on the other hand destroys sort order. The visualisation displays information about publicly traded companies at the US stock market. Colour coding is used to indicate changes in market value. The largest rectangles divide the visualisation into coarse categories such as energy or technology. Smaller rectangles represent individual companies, with market capitalisation as the size indicator. Tool tips display additional

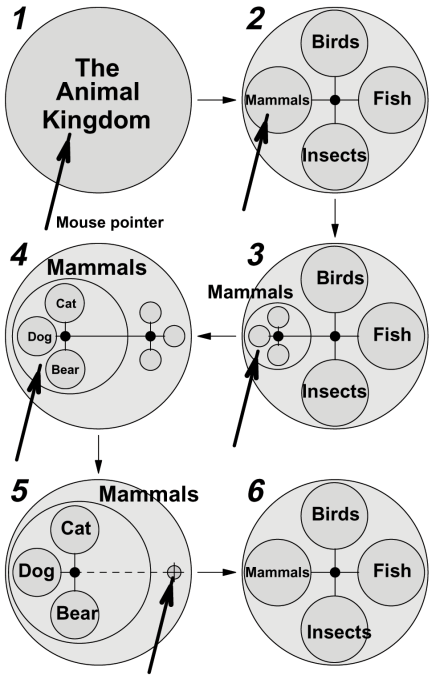


Figure 2.12: Tree browsing using Bubble Trees. Steps 1, 2, and 3 show the bursting of an opaque bubble, steps 3 and 4 show the expanding of the selected bubble, giving it more screen space. Steps 4 and 5 show the abstraction of unrelated bubbles into hyperbubbles. Steps 5 and 6 show the backtracking mechanism provided by the hyperbubble. [Extracted from Boardman [2000] under the terms of the ACM copyright. Copyright © by the Association for Computing Machinery, Inc.]

information such as the name of the company and the exact change in market value. Figure 2.13 shows a screen shot of a market map displaying the US stock market as on 4th December, 2006.

2.2.9 Cone Tree

The Cone Tree [Robertson et al., 1991] is a 3d conical representation of a tree. Each level of the hierarchy is displayed as a cone with the root as the apex and all children placed along the base. In case of occlusions, the visualisation can be rotated to reveal hidden items.

The tree can either be visualised vertically or horizontally, with the horizontal layout (also called a cam tree) allowing better labelling of nodes. When selecting a node, the visualisation is rotated and the node is brought to the front. Additionally, the path from the root node to the selected node is highlighted. Figure 2.14 shows an example of a cone tree.

2.2.10 XDU

XDU [Dykstra, 1991] is a graphical disc usage utility for the X windows system. It displays the contents of a Unix file system using stacked rectangles, similar to a tree map. Each level of the hierarchy of the file system is a separate column in the view. The root node is displayed on the left and child nodes are positioned on the right. The vertical space of each column is allocated proportional to the size of each subnode. Figure 2.15 shows the file system hierarchy of the Linux 2.6.26 kernel.

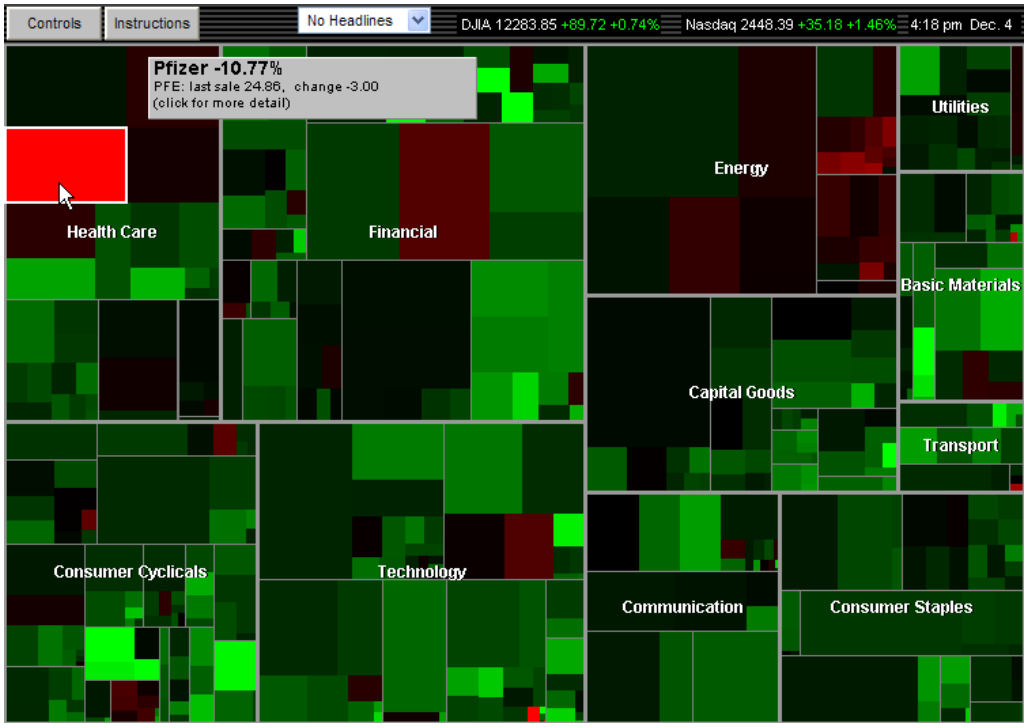


Figure 2.13: Market Map showing stock values, with red displaying falling values, green rising values on 4th December 2006. Hovering the mouse cursor over a node displays a tooltip with detailed information. [Screen shot taken from SmartMoney [2006].]

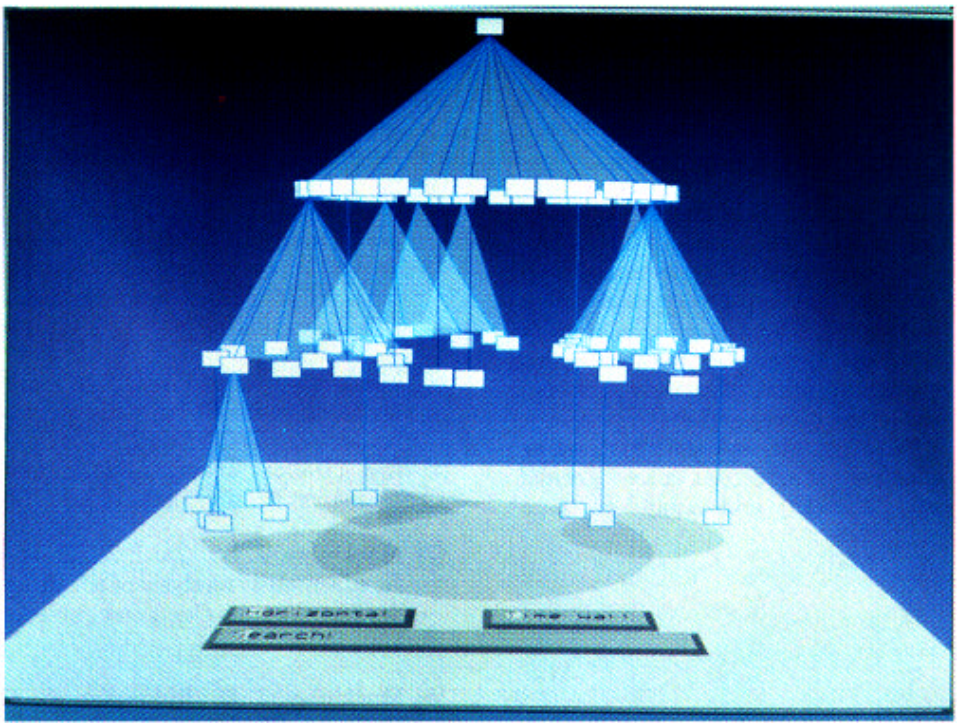


Figure 2.14: The Cone Tree displaying a 3d visualisation of a hierarchy. Each level of the hierarchy is displayed as a cone. The visualisation can be rotated to bring selected nodes to the front. [Extracted from Robertson et al. [1991] under the terms of the ACM copyright. Copyright © by the Association for Computing Machinery, Inc.]

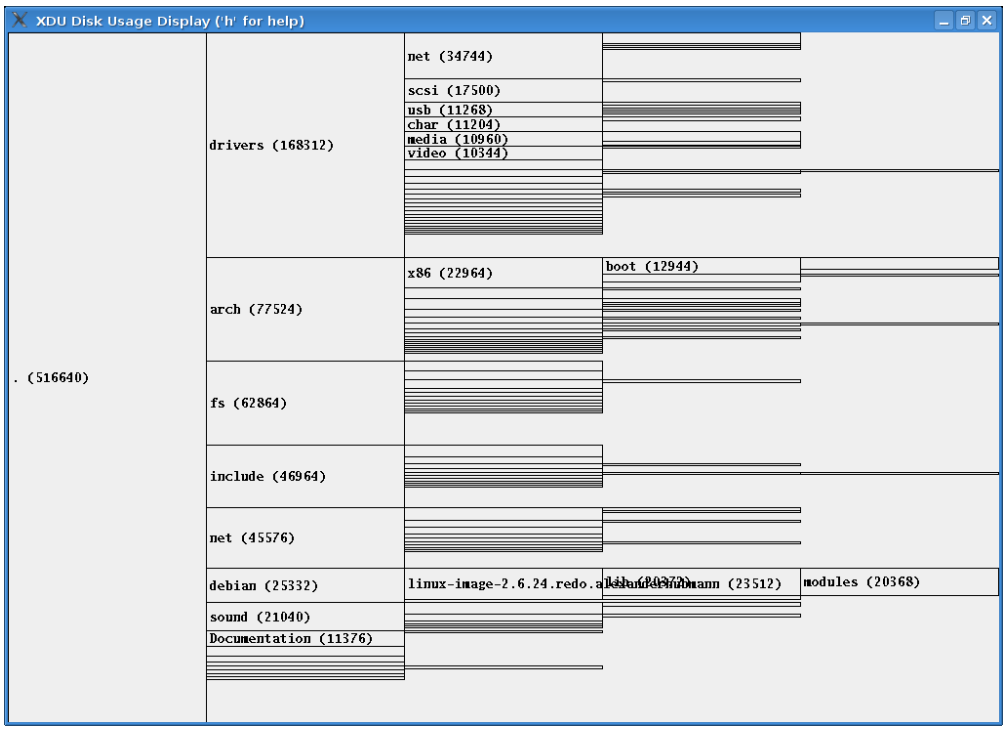


Figure 2.15: XDU showing the file system hierarchy of the Linux 2.6.24 kernel. Each level of the hierarchy is displayed in a separate column. The root node is displayed on the left, child nodes are positioned on the right.

2.2.11 Information Pyramids

Information Pyramids [Andrews et al., 1997; Andrews, 2002a] use three dimensions to visualise large hierarchies. A plateau represents the root node of the tree, subtrees are visualised using smaller plateaus on top of the root plateau. Files or documents are represented using separate icons. The size of each plateau is proportional to the size of its contents. Figure 2.16 shows an example visualisation using information pyramids. The pyramids grow upwards as the hierarchy is descended.

2.3 Networks

2.3.1 Harmony Local Map

The Harmony Local Map [Schipflinger, 1998] displays a dynamically generated graph depicting links between documents. Nodes represent documents, links are visualised as edges in the graph. Focus is put on the selected document, placing it in the centre of the screen. Links pointing to the document (incoming links) are displayed to the left of the selected document, links pointing from the document to other documents (outgoing links) are displayed to the right. Selecting another document puts it into focus and regenerates a new map for that document. Figure 2.17 shows an example of the Harmony Local Map, displaying links to and from the “grep” UNIX manual page.

2.3.2 Harmony 3D Local Map

The Harmony 3D Local Map [Wolf, 1996], which visualises both hierarchical structure and hyperlinks, extends the Harmony Information Landscape [Andrews et al., 1996]. The hierarchy is displayed horizontally in a two-dimensional landscape. Hyperlinks between documents are visualised using the vertical

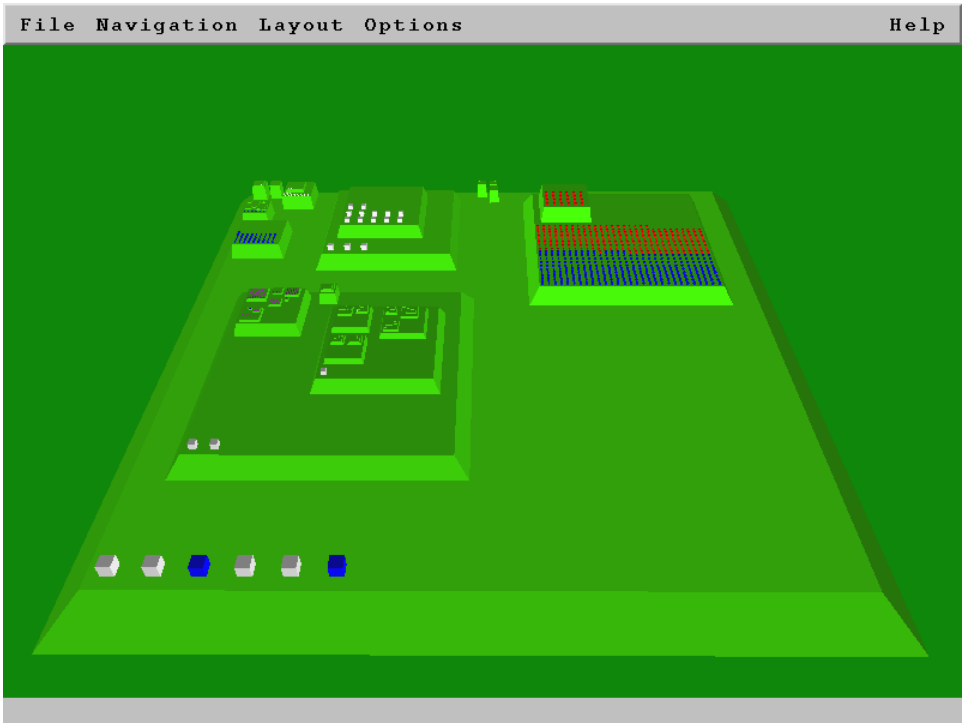


Figure 2.16: Information Pyramids using a 3d layout to visualise large hierarchies. Plateaus represent levels of the hierarchy, the pyramids are growing upwards as the hierarchy is descended. [Extracted from Andrews et al. [1997]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

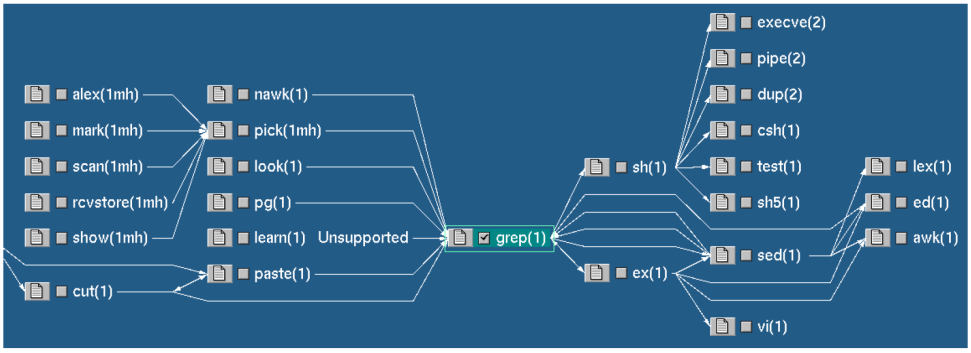


Figure 2.17: An example of the Harmony Local Map displaying the link neighbourhood for the "grep" UNIX manual page. The selected document is displayed in the centre, incoming links are displayed on the left, and outgoing links on the right. [Extracted from Andrews [2002b]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

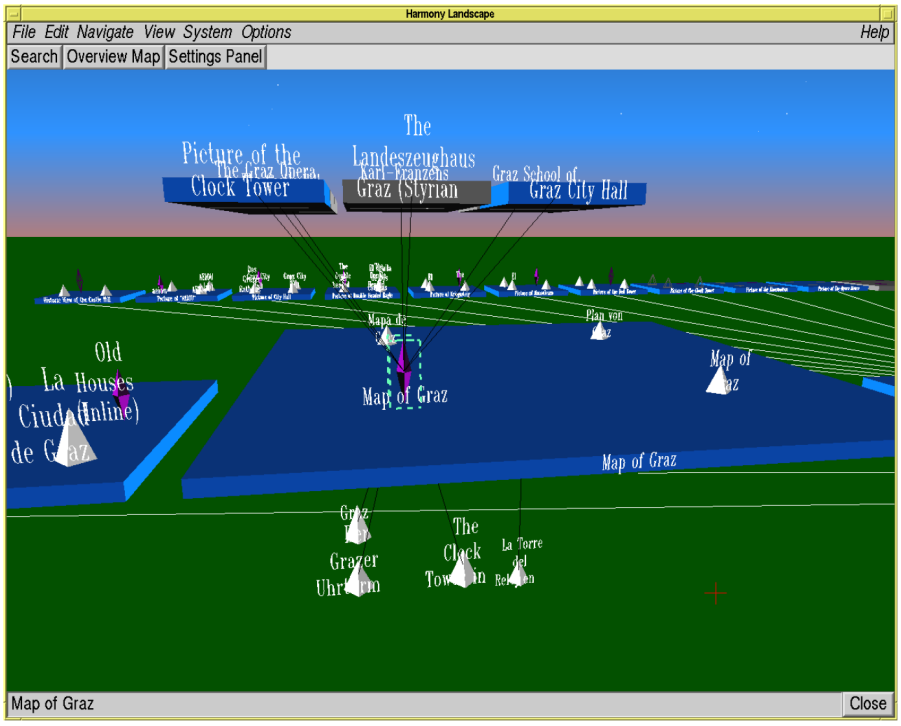


Figure 2.18: An example of the 3D Harmony Local Map displaying both hierarchical information and hyperlink relationships. Documents linking to the selected "Map of Graz" document are visualised below the document. Documents referenced by the selected document are placed above the document. [Extracted from Andrews [2002b]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

axis. Elements linking to the selected document (incoming links) are placed below the document, documents referenced by the selected document (outgoing links) are placed above the document. Figure 2.18 shows an example of a three dimensional map depicting links pointing to and from the document "Map of Graz".

2.3.3 HyperSpace (Narcissus)

Hyperspace [Wood et al., 1995] displays a 3d visualisation of web pages. Each document is visualised as a sphere, links between pages are displayed as edges between the spheres. The size of each sphere is determined by the number of links. Initially, the documents are placed at a random position. Using a spring model, spheres push each other away while links between spheres act as an attractive force. Figure 2.19 shows a ray-traced image of a map of over 750 pages.

2.4 Multidimensional Metadata

2.4.1 Parallel Coordinates

Parallel Coordinates [Inselberg and Dimsdale, 1990; InfoScope, 2008] is a system for visualising multi-dimensional data in a two-dimensional space. For each dimension in the n-dimensional space, a copy of the vertical axis in a Cartesian coordinate system is created. A point in the n-dimensional space is then assigned n points on the n axes in the two-dimensional space. The polygonal line connecting these points represents the point in the n-dimensional space. Thus, each point in the n-dimensional space is mapped to a polyline with n vertices.

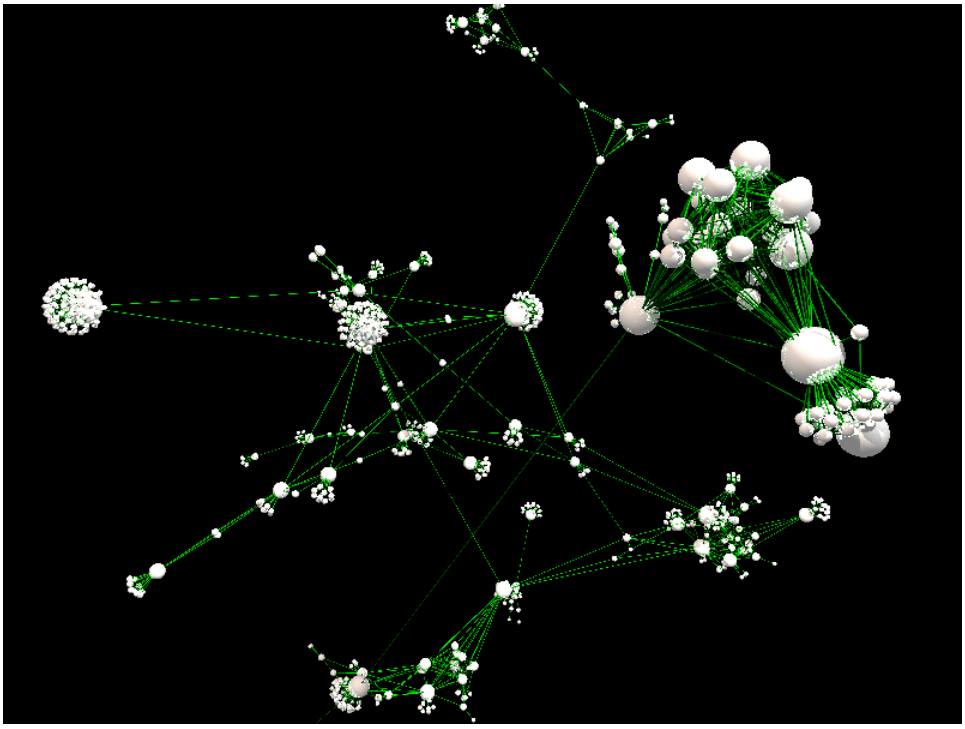


Figure 2.19: The HyperSpace visualisation showing a map of over 750 documents. Each document is represented using a sphere, links between documents are rendered as edges in the graph. [Extracted from Wood et al. [1995]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

By flattening the n-dimensional space to a two-dimensional visualisation, similar points can be easily spotted as they are represented by similar polylines. Figure 2.20 shows an example of a visualisation displaying 21 objects in an information space with 31 dimensions (using 31 parallel coordinates).

2.4.2 Table Lens

Rao and Card [1994] describe Table Lens, a visualisation technique for large tables. It uses a spreadsheet layout and provides a focus-plus-context view on the data, allowing the user to both see detailed information on selected objects, but at the same time keep the entire information structure in view. A commercial version is available from BusinessObjects (an SAP company) [BusinessObjects, 2008].

Using a fisheye distortion, the selected rows and columns in the spreadsheet are focused while the rest of the data is minimised. This way, the Table Lens visualisation can display up to 30 times more cells than a traditional spreadsheet. Additionally, the values in the minimised cells are converted into graphical representations (such as horizontal bars representing the values in the cells) to both exploit the economy in showing cell values this way and to help the user see patterns and features in the graphical rendering.

2.4.3 Attribute Explorer

The Attribute Explorer [Tweedie et al., 1994] is an interactive, graphical tool to visualise the relationships in multi-attribute data. The visualisation maps each attribute in the multi-dimensional data to a single-dimensional representation, similar to a histogram.

Figure 2.22 shows an example of the visualisation, displaying a set of houses along with their corresponding price, the number of bedrooms, and additional data. Each house appears once on each axis. For

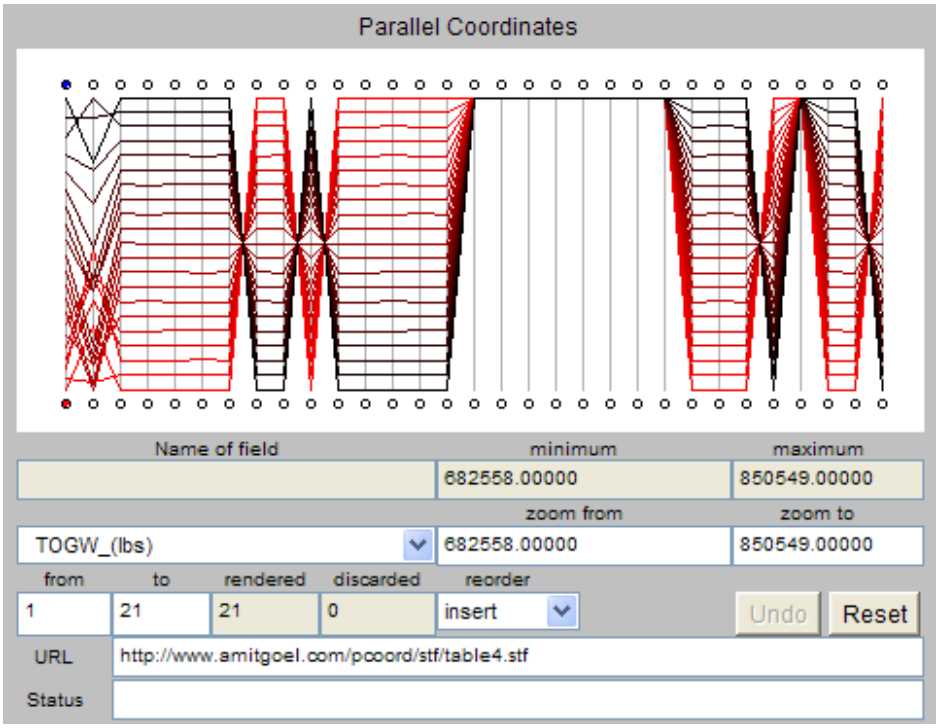


Figure 2.20: Parallel Coordinates displaying 21 objects on 31 dimensions. Data is taken from parameters of an aircraft design, representing values such as wing span, amount of fuel and cruise altitude. [Screen shot of Java Applet by Goel [2006].]

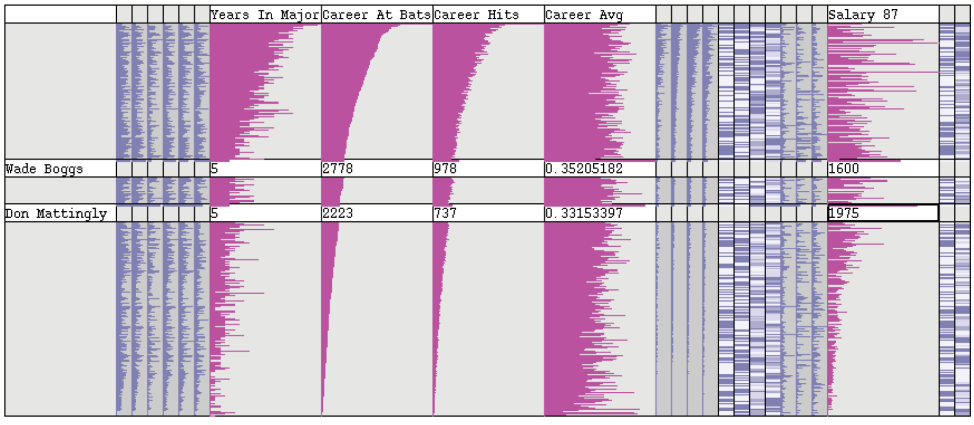


Figure 2.21: Table Lens displaying baseball statistics of 323 players by 23 variables. Selected rows and columns are focused while the rest is minimised. Values in minimised cells are converted into graphical representations (such as horizontal bars). [Extracted from Rao and Card [1994] under the terms of the ACM copyright. Copyright © by the Association for Computing Machinery, Inc.]

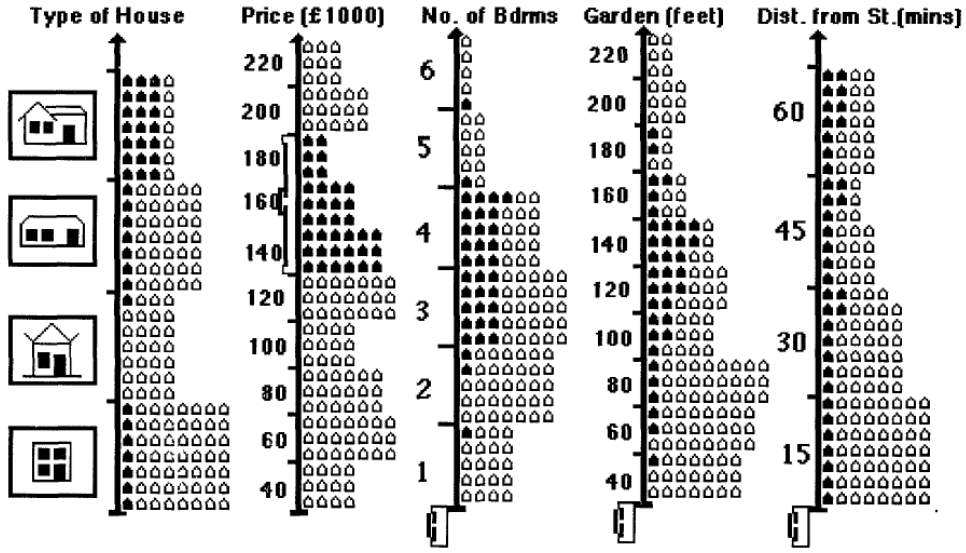


Figure 2.22: Attribute Explorer showing the distribution of selected houses across different attributes, such as price, number of bedrooms, and the size of the garden. Each attribute shows the distribution of houses as a histogram. Selecting items in one axis automatically highlights them on the other axes. [Extracted from Tweedie et al. [1994] under the terms of the ACM copyright. Copyright © by the Association for Computing Machinery, Inc.]

each attribute, the distribution of the data is shown. If the user selects specific items (in this case, houses in the range of £140,000 to £180,000), the items are automatically highlighted along the other axes.

2.5 Feature-Based Vector Spaces

Vector space analysis is often used to calculate and compare the similarity of documents. Each document is assigned a so called term vector, constructed from the list of unique words in the document collection. A stop word list is applied to the list, which removes words common to most documents, such as “a” and “the”. The value of each component in the vector (typically a value between 0 and 1) is proportional to the frequency of the word in the document. To measure the similarity between two documents, the scalar product of the two term vectors is calculated. The following visualisations lay out the documents corresponding to the calculated similarity.

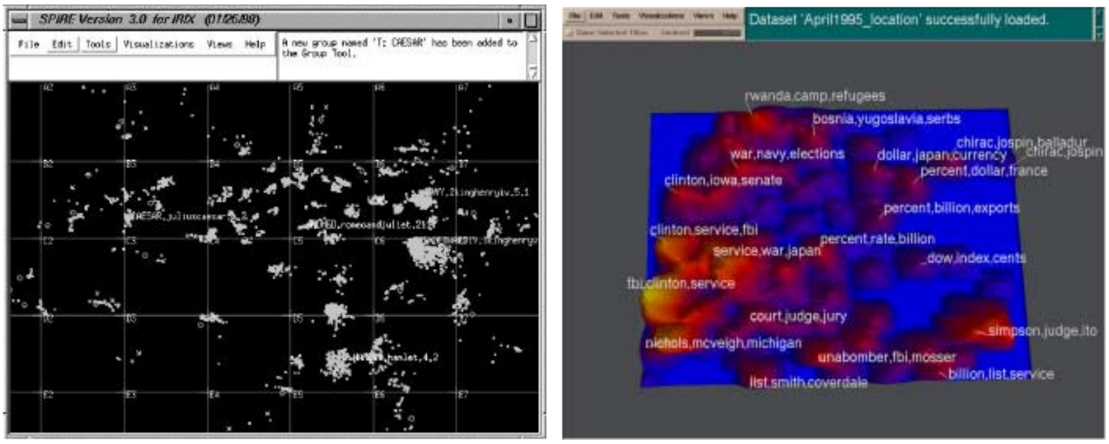
2.5.1 SPIRE Galaxy View and ThemeView

SPIRE’s Galaxy View [Hetzler et al., 1998; Thomas et al., 2001] visualises a collection of documents as a galaxy of stars. Similar documents are placed close together, dissimilar documents are placed farther apart. ThemeView visualises documents in a thematic landscape. More significant concepts form hills (higher peaks), colour coding visualises the calculated height field.

The text analysis engine used in both Galaxy View and ThemeView uses hierarchical or k-means clustering algorithms to process the data sets. Labels are placed on higher peaks, listing the most frequent themes of documents in the cluster. Figure 2.23 shows examples for both Galaxy View and ThemeView.

2.5.2 VisIslands

VisIslands [Sabol, 2001; Andrews et al., 2001], similar to SPIRE’s ThemeView, uses the xFIND [Gütl, 2000] search engine to display dynamic, thematic clusters of search results in real time. The results are



(a) **(b)**

Figure 2.23: SPIRE's Galaxy View and ThemeView. (a) shows the Galaxy View, visualising a collection of documents as a galaxy of stars. (b) shows ThemeView, which places documents in a thematic landscape. More significant concepts form higher peaks, colour coding visualises the calculated height field. [(a) extracted from Hetzler et al. [1998], (b) extracted from Thomas et al. [2001]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

first pre-clustered, the centroids are then randomly distributed in the display area. All documents belonging to a cluster are arranged in a ring around the previously placed centroid. A force-directed placement algorithm in which similar documents attract each other fine-tunes the position of the documents.

After the system has stabilised, a height field is constructed, with dense areas leading to higher ground (peaks) and sparse areas to lower ground (valleys). By assigning different colours to different heights, the resulting image resembles a map with ocean and islands. Figure 2.24 shows an example of a calculated VisIslands map.

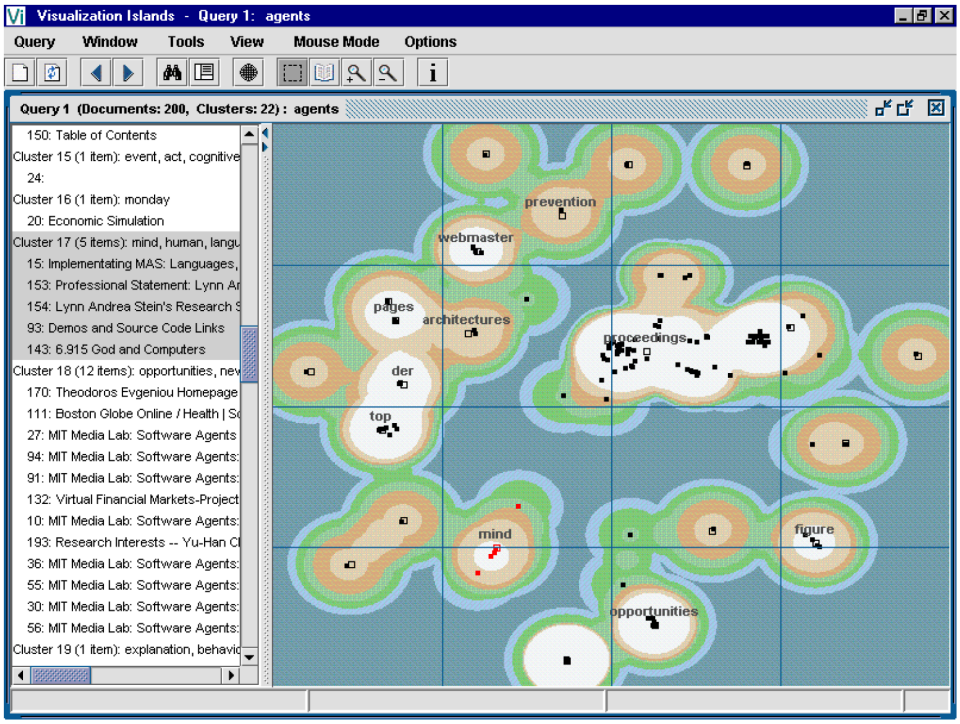


Figure 2.24: VisIslands displaying a landscape of 22 clusters, calculated from 200 documents. The documents in the selected cluster are highlighted both in the map and the list of documents to the left. [Extracted from Andrews et al. [2001]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

Chapter 3

Email Readers

"Two of the cruellest, most primitive punishments our town deals out to those who fall from favour are the empty mailbox and the silent telephone."

[Hedda Hopper]

Email has become an ubiquitous communication medium, not only for sending personal messages, but newsletters, reminders, shopping receipts and much more. Web mail, at first only a less-than-ideal alternative for checking emails when on the move, has improved dramatically. For many users, web mail nowadays is the only way to access their mail messages, pushing traditional desktop mail clients to the enterprise. Traditional mail applications developed from only providing access to mail messages to personal information managers, including to-do lists, calendars, and contacts management.

The following sections provide a quick overview of common mail clients: Microsoft Outlook Express (Section 3.1) is included in every Microsoft Window installation. The more professional Microsoft Outlook (Section 3.2) is part of the Microsoft Office package, extending the mail client to a personal information manager. Mozilla Thunderbird (Section 3.3) is the best known open source mail client. Linux desktop environments such as KDE and GNOME also include their own mail clients: KMail (Section 3.4) in KDE and Evolution (Section 3.5) in GNOME. GNUMail.app (Section 3.6) is another example of an open source mail client, including a Thread Arcs (see Chapter 5) like visualisation. Finally, Google Mail (Section 3.7) and Windows Live Hotmail (Section 3.8) are presented as examples of modern web mail clients.

3.1 Microsoft Outlook Express and Windows Mail

Microsoft Outlook Express [Microsoft, 2007b; Wikipedia, 2008c] is an email and news client that has been included in Microsoft Windows since Windows 95. In Windows Vista, it has been replaced by Windows Mail. It features the basic three-pane layout (list of folders, list of messages, message details) present in almost any modern email client. The client can manage multiple email accounts and supports POP3 and IMAP protocols.

Windows Live Mail adds additional features such as a tight integration with the Windows Live Messenger, extended junk mail and fraudulent/phishing mail controls, displaying a yellow bar above suspicious emails. Quick views allow the user to display specific categories of email messages from different accounts in a single list. Windows Live Mail also provides usability enhancements such as the possibility to send high resolution images where only thumbnails of the sent pictures are included in the email itself which link to a private web location with the full resolution images.

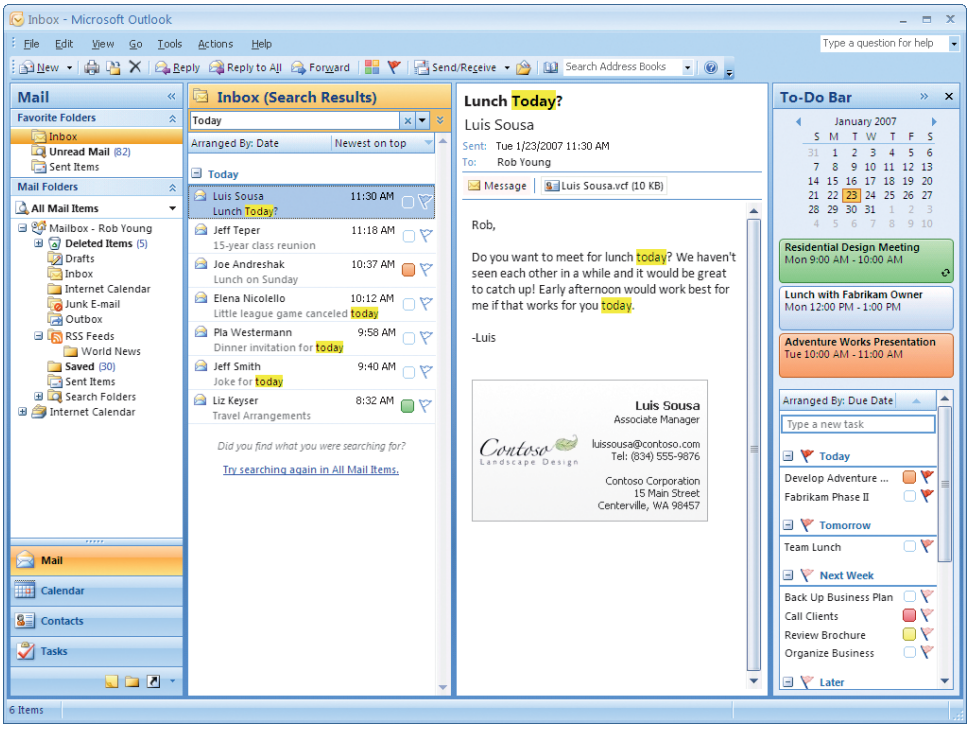


Figure 3.1: Microsoft Outlook 2007 showing the folders list, the inbox list, a preview of an attached file, the calendar and the to-do list. [Screen shot from Microsoft [2008b].]

3.2 Microsoft Outlook

Microsoft Outlook [Microsoft, 2007a; Wikipedia, 2007] is an email and personal information manager, distributed as part of the Microsoft Office package. In integrates access to email, contacts, shared calendars and to-do lists into a unified user interface.

Outlook 2007 includes instant search across email messages, attachments, calendar entries, contacts and tasks. Email messages can be flagged as tasks, task list entries can be integrated into the calendar view. Collaboration like shared calendars and contacts are provided by the Microsoft Exchange Server, which also provides features such as push-email to integrate hand-held devices. Figure 3.1 shows the Outlook 2007 user interface, showing email messages, the calendar and the to-do list.

3.3 Mozilla Thunderbird

Mozilla Thunderbird [Mozilla, 2006b,c] is an open-source email client developed by the Mozilla Foundation. It evolved from the Mozilla Application Suite, which was based on the source code of Netscape Communicator, released by the Netscape Communications Corporation in 1998 (although the current codebase of Mozilla is a complete re-write). A detailed history of the Mozilla Application Suite can be found in Wikipedia [2008b].

Mozilla Thunderbird is based on a cross-platform application framework, making it available on a wide range of platforms, including Microsoft Windows, Mac OS X, and Linux. Additional features are provided by so-called add-ons (see Chapter 6), which can be downloaded and installed.

Like most other email clients, Thunderbird supports the standard mail protocols POP3, IMAP and SMTP. Features include the support of multiple accounts, both email and newsgroup. Searching, virtual folders, message and junk filtering simplify message management. Figure 3.2 shows the standard user interface layout with a list of all accounts and folders on the left (1), a list of all messages in the selected

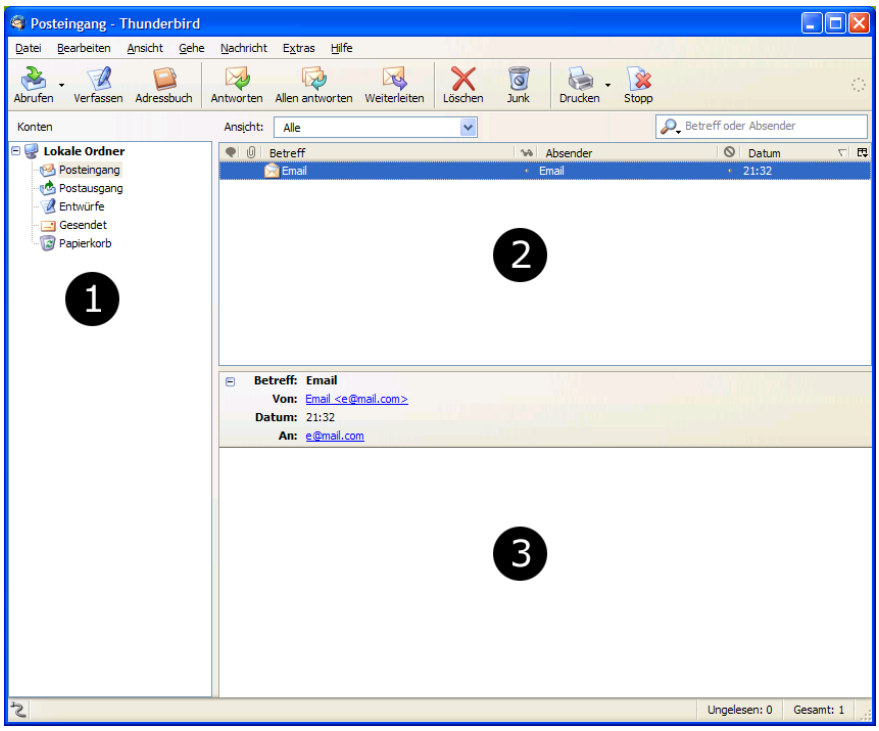


Figure 3.2: Mozilla Thunderbird user interface: list of all accounts and folders (1), list of messages in selected folder (2), and selected message (3).

folder (2), and the selected message itself on the right (3).

3.4 KMail

KMail [Project, 2006] is the email client of the K Desktop Environment. It supports the standard email protocols IMAP, POP3 and SMTP. Emails can be encrypted and signed using OpenPGP. Its editor supports spell-checking and undo/redo. The mail client can handle multiple accounts and identities, it has a threaded messages view, and quoted text can be highlighted using different colours. KMail supports virtual folders, which display a list of messages based on a search query. Messages can be composed either in plain text format or using HTML. Its anti spam wizard helps users sort out unwanted messages.

KMail can be seen in Figure 3.3. It shows the standard three-pane view common to most email clients. The left column shows all configured accounts and folders of the user, the right column shows the list of messages in the selected folder and the text of the selected message.

3.5 Evolution

Evolution [GNOME, 2008a] is the mail client of the GNOME [GNOME, 2008b] project. In addition to POP and IMAP, it integrates with Novell Groupwise and the Microsoft Exchange Server. It supports signing and encryption of messages using either GPG or S/MIME. Evolution also provides a calendar and integrates with the instant messenger Pidgin [Pidgin, 2008]. Search queries, even across different mail accounts, can be stored as dynamic folders. The search result can be extended to include the whole thread of matching messages, even if the other messages do not match the search criteria.

Evolution is not only a simple mail client, but also includes a calendar (which can be synchronised with Novell Groupwise and Microsoft Exchange), a contacts manager (which can retrieve its data from

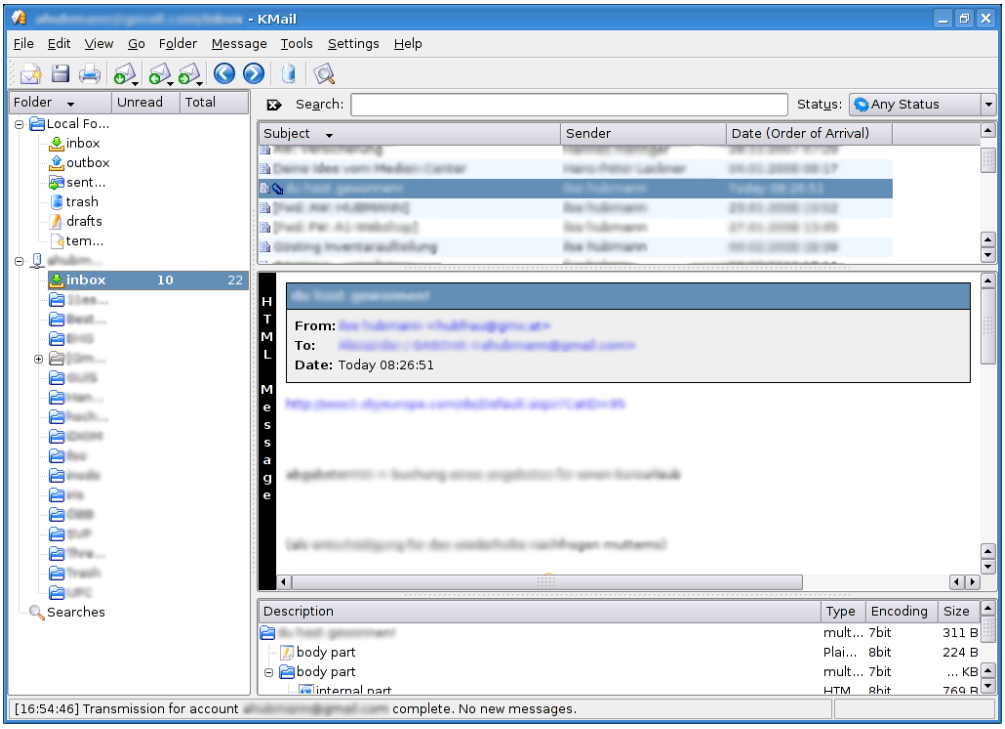


Figure 3.3: The KMail user interface with the list of accounts (left), the list of messages in the current folder (top right), and the selected message (middle right).

an LDAP server), and a task list. All data managed in Evolution can also be synchronised to hand-held devices.

Figure 3.4 shows the Evolution client. It shows the standard layout with the list of accounts and folders on the left, the list of messages on top, and the selected message at the bottom. To switch between the mail client, the calendar, the contacts, and the task list, the program provides a set of buttons below the account list.

3.6 GNUMail.app

The GNUMail.app [GNUMail.app, 2006] email client is built on top of GNUStep [GNUStep, 2008] and Apple Cocoa [Apple, 2008] and runs on Apple Mac OS X, Linux, and BSD and is licensed under the GPL. It supports the standard email protocols POP3 and IMAP, uses the standard 3-pane layout displaying the list of accounts, the list of messages, and the selected message itself. An address manager is also integrated.

Similar to other mail clients, emails can automatically be sorted using filters and the client includes a spam filtering module. GNUMail.app includes advances features such as mailing list detection giving the user the option to reply to the whole list or only to the sender of the message.

A basic Thread Arcs visualisation as described in Section 5 was successfully integrated into GNUMail.app. Figure 3.5 shows the basic layout of the client and the Thread Arcs visualisation.

3.7 Google Mail

Google Mail or GMail [Google, 2006] is included in this list of email clients as an example of an advanced web-based email service. Although it is browser-based, it tries to mimic a desktop email-client

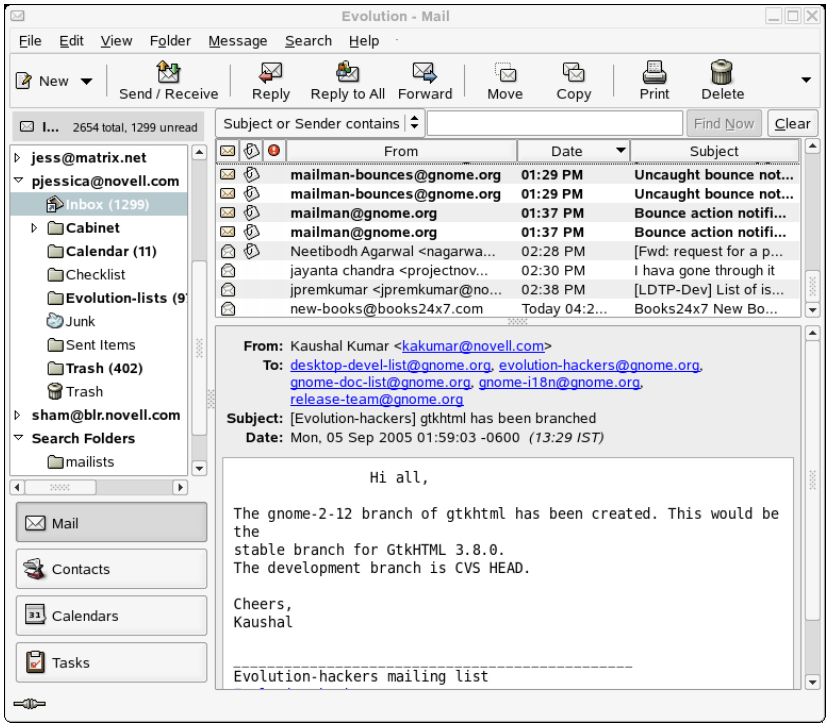


Figure 3.4: The GNOME Evolution Mail Client with a list of accounts and folders on the left, a message list on top and the selected message at the bottom. Evolution also includes a calendar, a contacts manager and a task list, which can be accessed using a set of buttons below the account list. [Screen shot from GNOME [2008a].]

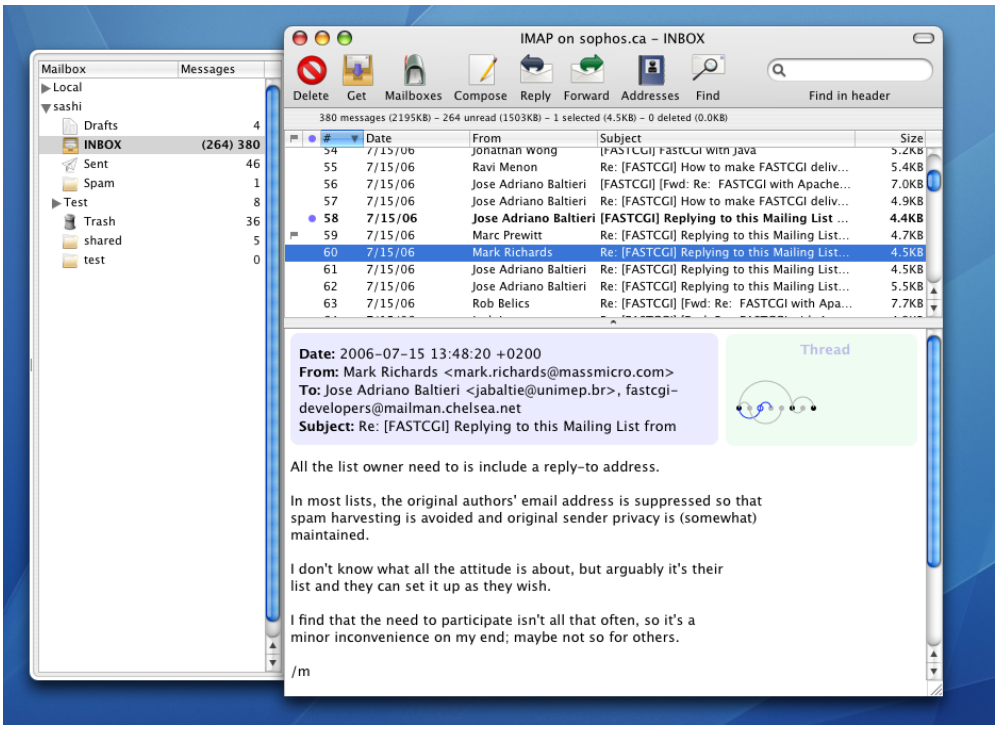


Figure 3.5: The GNUMail.app running on Mac OS X. A basic version of the Thread Arcs visualisation is displayed next to the header section. [Screen shot from GNUMail.app [2006].]

as far as possible. By using advanced techniques such as dynamic HTML, JavaScript and AJAX, it eliminates page reloads almost entirely. This makes it a prime example of a Web 2.0 application. When introduced on April 1, 2004, it stood out from other web-based freemail services by offering one gigabyte of storage. Similar to the Google homepage itself, text-based advertisements are displayed in the browser window. According to Wikipedia [2008a], the main features of Google Mail are:

- *Conversation Views* provide context for each sent and received email by displaying all messages that are part of the conversation (the original message, along with all replies) when the user selects a message. Certain operations can also be applied to whole conversations, such as delete or move.
- *Labels*, similar to folders, allow the user to categorise all messages. Unlike folders, each message can be assigned multiple labels.
- *Searching* allows the user to find any email, based on a large number of criteria, such as full text search, the message's From, To, and Subject fields, assigned labels, the message's date, etc. Indeed, "search, don't sort" is a slogan for GMail.
- *Contacts* are managed automatically by GMail. All addresses of received and sent mails are stored in the address book. When composing a new message, auto-completion suggests possible recipients.
- *JavaScript Interface* The heavy use of JavaScript and dynamic HTML makes the GMail interface very responsive. To support older browsers, a "basic HTML view" is provided.
- *POP3 and IMAP* allows the use of GMail by any standard desktop-based email client.
- *Addresses* GMail ignores any dots in email addresses, so that `address@gmail.com` is the same as `add.ress@gmail.com`. Additionally, plus-addressing is supported, allowing for the creation of additional email addresses in the form `address+tag@gmail.com`.
- *Anti-virus scanning* of all incoming and outgoing mails reduces the risk of infection by viruses. Additionally, it is not possible to send executable files as attachments.
- *Spam filtering* helps the user to move unwanted mails to the spam folder.
- *Rich text formatting* allows users to compose messages in HTML format using a What You See Is What You Get editor.

Figure 3.6 shows the main GMail interface. Viewing a selected message automatically shows all messages that are part of the same conversation. The user can display the content of any message in this conversation by clicking on the message header.

3.8 Windows Live Hotmail

Windows Live Hotmail (previously called Hotmail) was launched in 1996. Today, it features 5 GB of email space, advanced search including keywords (e.g. `from:` to limit the search to the name of the sender), a spell checker, and much more.

Similar to Google Mail and Google Talk, Windows Live Hotmail is integrated with Windows Live Messenger and displays the online status of the user's contacts. The user interface provides advanced features such as drag-and-drop of messages to folders and context menus.

Security warnings display red markers for phishing emails or messages containing fraudulent content. These messages are automatically blocked. Yellow warnings indicate messages from senders not on the user's contacts list. For those messages, any attachments, pictures, and links are automatically blocked. Figure 3.7 shows Windows Live Hotmail with the folders list on the left and the inbox list on the right.

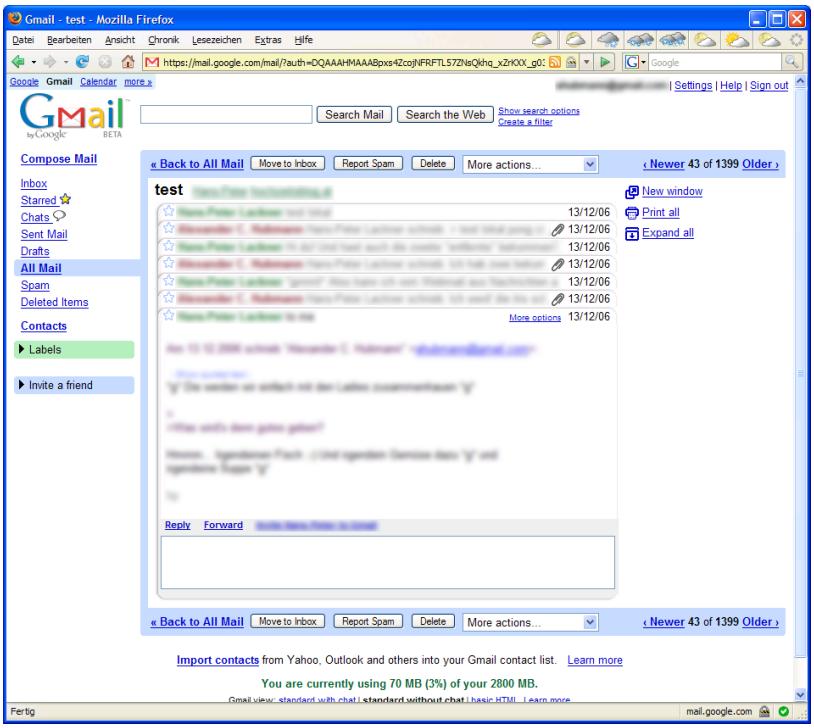


Figure 3.6: The GMail user interface automatically displays all messages belonging to a conversation alongside the selected message to give context. Applying a label to a message also adds the label to other messages in the same conversation.

Figure 3.7: Microsoft Live Hotmail showing the standard layout with the list of folders on the left and the inbox list on the right. Messages are moved to folders using drag-and-drop, similar to desktop based mail clients. [Screen shot from Microsoft [2008a].]

3.9 Conclusion

Traditional desktop email clients and web mail applications both provide access to email messages, although they greatly differ in their target user group and features. Since the invention of AJAX and dynamic web applications, web mail clients feel more and more like desktop applications, with the difference that they are accessible from any computer, from anywhere. On the other hand, users lose physical control over their email, bringing up issues such as backup, security, and privacy.

At first, web mail was only a secondary means to access email, when the user was away from the desktop but had to check email. Compared to desktop clients, it provided far less features. Fast forward to today: for more and more users, web mail is the only access to their emails. Most providers offer practically unlimited storage for normal users, include automatic virus scanners, and all this (for the most part) for free.

Web mail providers such as Google, Yahoo, and Microsoft at first only provided browser-based, online access to the user's mailbox. With added features such as POP3 and IMAP access, the line between pure web mail providers and traditional email providers blurs, as users can access their mail both via traditional mail clients and web mail interfaces.

Although web-based solutions are becoming more and more desktop like, traditional mail clients will not disappear anytime soon. Features such as managing multiple email accounts across different servers and different providers, newsgroup access, integrated encryption and signing of messages, and of course offline access to email can generally be found only in desktop applications.

Chapter 4

Email Visualisation

> *"In particular, we need to move beyond visualizing the easy features of email and help people better manage their tasks and relationships."*
>
> [Rohall et al. [2001]]

Mail clients typically provide a list of accounts, a list of folders in an account, a list of messages in a selected folder, and the selected message itself. Email visualisations try to improve the user experience by either providing a separate application that analyses a set of messages or by adding a visualisation to the email client, providing the user with additional input or output user interface elements.

ReMail (see Section 4.1) suggests a new client user interface, focusing on email conversations, to improve the usability of the mail program. eArchivarius (see Section 4.2), on the other hand, is a separate application supplementing a normal client and supporting the user in accessing large archives of emails. EzMail (see Section 4.3) is an information visualisation tool that runs parallel to a normal mail client, providing several different visualisations. The Email Mining Toolkit (see Section 4.4) analyses the social structure of multiple users by analysing their inboxes. faMailiar (see Section 4.5) analyses the content of emails and calculates contact and message intimacy properties using natural language processing techniques. Themail (see Section 4.6) analyses a users inbox and extracts the most relevant terms used in mails over time. Similar to ReMail, Email Conversations (see Section 4.7) focuses on email conversations and provides the user with a visualisation displaying the thread of the selected email.

4.1 ReMail

ReMail (described in Rohall et al. [2001]; Rohall [2003]; Kerr [2003a]; Rohall et al. [2004]; Kerr and Wilcox [2004]; Gruen et al. [2004]) is a prototype email client developed by IBM that went through several iterations until a first working prototype emerged in summer 2001. It tries to add context to email messages, as the messages themselves provide little context on their own. ReMail displays annotations, reminder, to-dos, the online status of authors, the message source, thread size and several additional pieces of information alongside the normal email display. Figure 4.1 shows a prototype of the ReMail client.

Pivoting

By pivoting, the user is able to change the sort order of the displayed messages while keeping the selected message in focus. Even when changing the view to, say, the list of all messages in the thread, the current message keeps its focus. The user can then change the focus to a related message (by clicking on it) and pivot again to other views.

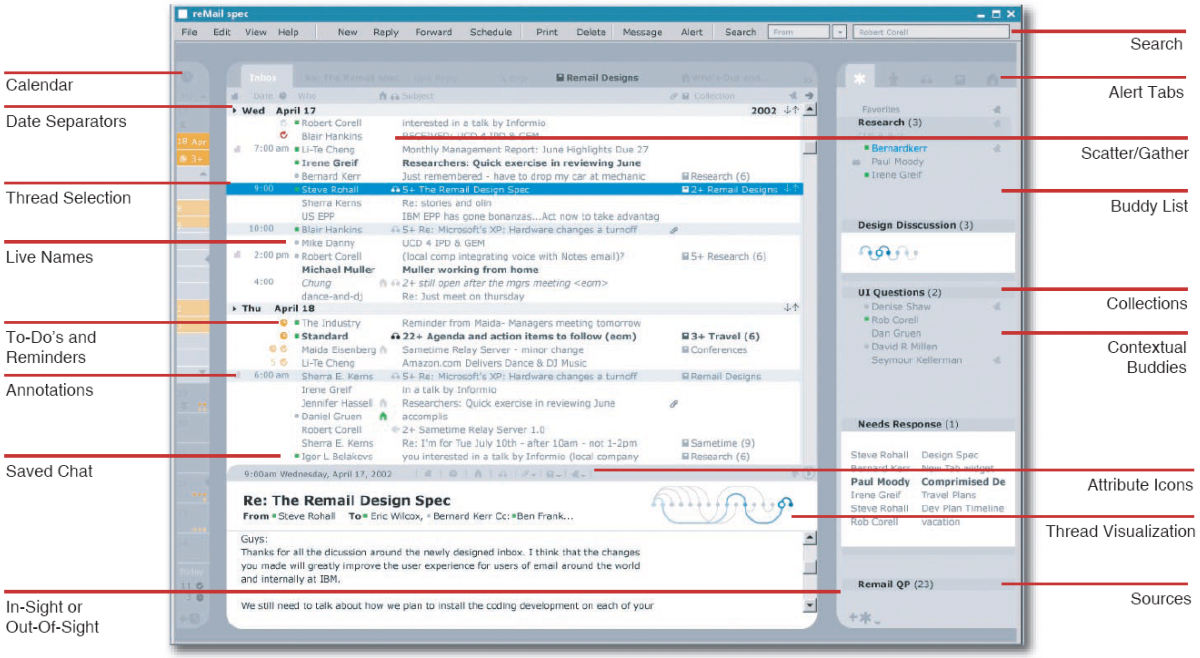


Figure 4.1: The ReMail client showing the inbox list. [Extracted from Kerr and Wilcox [2004]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

Threads

A thread is a conversation created by the email reply function, forming a group of related messages. If a user selects a message in the ReMail interface, messages belonging to the same thread are automatically highlighted. The Thread Arcs visualisation (see Section 5) provides additional information about the conversation, giving a compact overview over all messages in the thread. It provides context to the currently displayed email without taking up unnecessary screen space. A separate tab on the right displays more information about the conversation, such as the time span of the thread or a list of all participants.

Chat

Integrated chat functionality provides an additional means to communicate. Any name field in the user interface automatically visualises the online status of the particular person. The name can also be used to start a new chat with the person. Chat messages can be organised just like normal email messages. A separate buddy list is included in the interface as an additional tab, a temporary list is automatically created from the list of recipients when viewing an email message.

Annotations

Email messages can be marked using annotations, similar to post-it notes, which are displayed as small coloured icons next to the message. Annotations can contain additional notes, and the inbox list can be filtered to only display annotated messages. Additionally, annotations can be read without opening the corresponding email messages.

Calendar

As a great deal of work is received and delegated in email, ReMail provides convenient methods to assign calendar markings to email messages. Dragging a message onto the calendar tab automatically creates a

new reminder in the calendar, which also automatically appears in the inbox at the specified date. Date information extracted from the email body is automatically prefilled in the calendar dialogue. A message can also be marked as a to-do item which automatically puts it into the to-do list. Similar to reminders, appointments (with additional time information) can also be created from email messages. Contrary to reminders, they are only visible in the calendar view. The calendar view is also coupled to the inbox view. Selecting a date in the calendar automatically scrolls the inbox to show messages received on this date.

Collections

Collections can be used to group email messages in manner similar to folders. They enhance the folder idea, since a single message can exist in multiple collections, whereas in traditional email clients a message is always associated with a single folder. To avoid clutter in the inbox, collections can be set to be displayed out of sight, i.e. the messages are moved from the inbox similar to normal folders. Setting a collection to be in sight keeps all messages in the inbox and marks them with an icon and a label for the collection. The collections tab lists all collections of the user and also highlights all collections the selected email belongs to. To navigate messages, the user can also pivot to all messages of a collection by clicking on a collection icon in the inbox list. Users can also set up automatic collections according to certain filter criteria. In contrast to normal mail clients and filters, messages stay in the inbox and are not moved to subfolders, keeping them in sight.

Scatter and Gather

In older email clients, messages in the inbox are sorted by date, separating messages of the same thread by unrelated messages. ReMail gathers read messages from the same thread and only displays the last message of a thread in the inbox view. The user can view all messages of the thread by double clicking on the message, which pivots to the list tab, which only lists messages from the selected thread.

4.2 eArchivarius

eArchivarius [Leuski et al., 2003; Leuski, 2006] is a system for accessing email archives. It provides a way to retrieve stored messages and make sense of the context they appear in. By combining ranked retrieval with cluster-based and time-based navigation, eArchivarius helps users to find relevant information.

Using a normal topic-oriented search, users end up with a list of relevant messages. Unfortunately, these messages are displayed out of context, so the user has to follow the relevant threads of conversation to get more information. In addition to providing this thread-based context, eArchivarius also displays a cluster-based visualisation that groups similar messages or people. The clusters are visualised as spheres floating in space, their proximity is based on a user-definable similarity function. This helps the user to choose the right similarity function depending on the type of context to be explored.

4.3 EzMail

EzMail [Samiei et al., 2004] is an email visualisation tool which provides the user with a multi-view interface running in parallel to the default email client. Messages are grouped and visualised as parts of conversational threads. The visualisation shows messages in detail and provides a contextual overview. The main interface is composed of five interconnected views, providing a focus-plus-context visualisation:

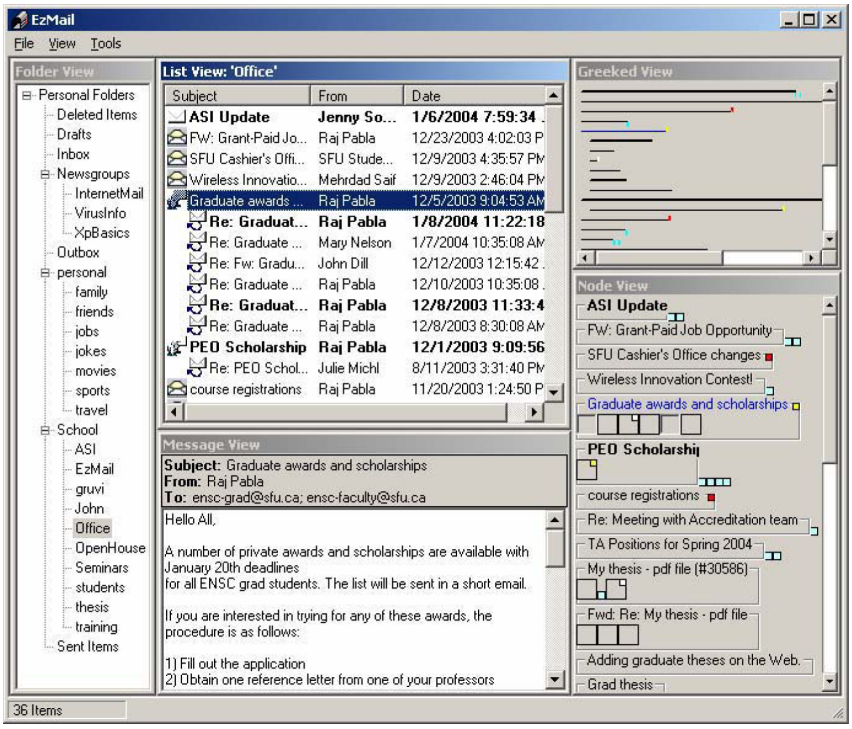


Figure 4.2: The EzMail visualisation with the folder view listing all folders, the list view displaying all messages in the selected folder, the message view showing the selected message and its headers, the greeked view providing an overview of all messages in the list view and the node view giving a detailed overview over all threads in the list view. [Extracted from Samiei et al. [2004]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

- *Folder View:* The folder view lists all personal email folders, similar to a typical email client.
- *List View:* The list view provides a list of all messages in the selected folder in the folder view. Emails are grouped by thread (displayed as a tree) and sorted by date.
- *Message View:* The message view displays the main content and headers of the selected email.
- *Greeked View:* The greeked view is a high-level overview consisting of horizontal lines which represent the relative sizes of email messages. Replies are indented the same way as in the list view. Unread messages are highlighted using thicker lines.
- *Node View:* A graphical view on the messages displayed in the list view is provided in the node view. Messages are clustered based on thread topic and are visually grouped by an enclosing frame. A single message or the first (root) message of a thread is shown as a frame node, displayed as the caption of the frame. Further replies in the thread are added as box nodes inside the frame element. Attachments, annotations and further meta data can be added to the visualisation using coloured lines and squares.

Two additional views are provided as pop-ups. The thread view displays all messages from a thread regardless of which folder they are stored in. This provides context and history for a selected message. The sender view selects all messages to or from a given person. Figure 4.2 shows the main interface of EzMail with the greeked view on the top right and the node view below.

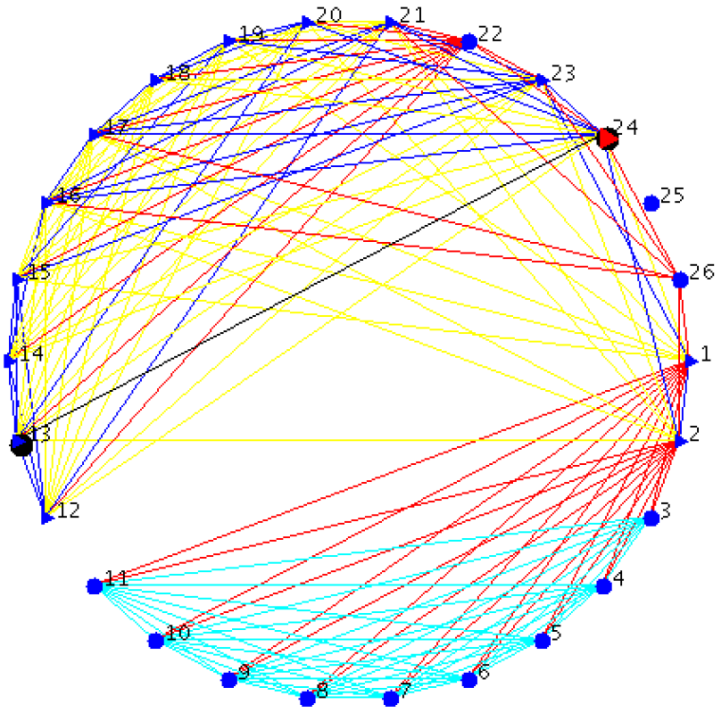


Figure 4.3: The Email Mining Toolkit (EMT) showing several different cliques and their connections (common members). [Extracted from Li et al. [2004] under the terms of the ACM copyright. Copyright © by the Association for Computing Machinery, Inc.]

4.4 Social Networks

The Email Mining Toolkit (EMT) [Li et al., 2004] analyses email archives and shows relationships between users and groups of users. To compute behaviour profiles or models of email accounts, 15 different features and models are included in EMT. The message table classifies individual emails using machine learning subsystems. A usage histogram shows the typical daily behaviour of an email user. Similar users are identified as groups of users who act in similar ways. The recipient frequency analyses typical communications with a user.

Models for group behaviour include the enclave clique which shows groups of users who frequently exchange messages between each other. The user clique on the other hand displays a set of correspondents a particular user emails as a group. Email flows reveal how a single message kicks off other messages to start new conversations. Another interesting statistical display is the average communication time which shows the response time to email messages grouped by correspondents.

EMT computes cliques based on data either from a single email archive or from the email archives of a group of users. When looking at a single email archive, user cliques can be constructed. By including the archives of more users, enclave cliques can be computed. These cliques reveal social groups that can be constructed from the email messages exchanged by a group of users. Figure 4.3 shows several cliques and their connections. A link between two nodes (two cliques) represents common members, i.e. users who belong to both cliques. Colours represent the percentage of users the two cliques share.

An email flow starts at a single message and follows its path to different correspondents. This path can be constructed by looking at the subject, common attachments, and similar content of later messages to infer the thread of the conversation. Figure 4.4a shows such an email flow. A node represents a sender or a recipient, exchanged messages between two users are displayed as edges in the graph. Colouring indicates either one-way or two-way communication between those users. Time and order of messages is shown using concentric rings. The starting point (the first message) is displayed in the middle, follow-

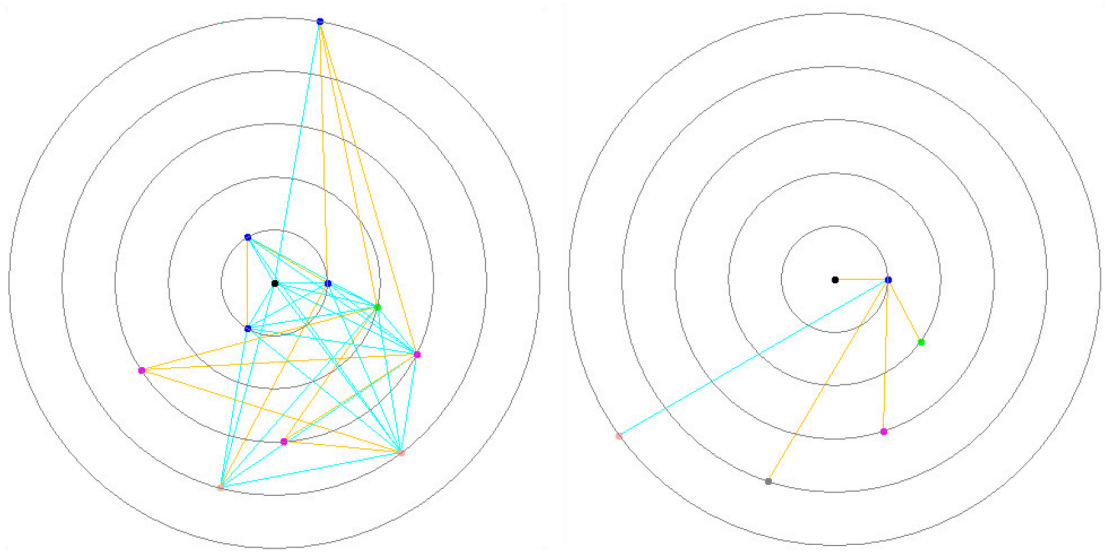


Figure 4.4: The Email Mining Toolkit (EMT) showing different email flows, with nodes representing senders and recipients and messages displayed as edges in the graph. (a) shows a normal email flow, (b) shows an email flow visualisation of a spambot, where a single account is creating multiple emails to many users in a short time. [Extracted from Li et al. [2004] under the terms of the ACM copyright. Copyright © by the Association for Computing Machinery, Inc.]

ing messages are drawn on concentric rings around the first message with the newest messages on the outermost ring.

Figure 4.4b shows another email flow, where a single email account created copies of the same email and sent them to many users in a short time. The pattern is obviously different to normal human communication patterns as seen in the previous figure which could indicate some sort of automated process creating the messages, for example a spambot or a secretary posting to a distribution list.

4.5 faMailiar

faMailiar [Mandic and Kerne, 2004, 2005] is a novel email visualisation that emphasises the personal character of email. It defines intimacy in email in two different categories: contact intimacy and message intimacy. Contact intimacy is a parameter a user can assign to email contacts. By categorising contacts into one of five intimacy categories, the level of intimacy between the user and the contact is determined. Message intimacy is a calculated parameter, based on message data using information retrieval techniques (intimate and anti-intimate keywords).

Both contact intimacy and message intimacy are encoded in a small icon. Using colour hue and shape, the five contact intimacy categories are displayed. Messages from the most intimate group are visualised in a warm yellow colour, the most un-intimate in cold grey. Additionally, for incoming messages, the shape of the icon representing the message is dependent on the intimacy category, ranging from a triangle for the most intimate group to a circle for the least intimate group. Messages sent by the user are always displayed as stars, but are appropriately coloured. The colour value of the icons is proportional to the message intimacy, making more intimate messages brighter and less intimate messages darker. Figure 4.5 shows an example of icons for the five different contact intimacy groups. The top row shows icons for sent messages, the bottom row for received messages.

These icons are laid out on a two-dimensional grid, similar to a calendar, providing daily, weekly and

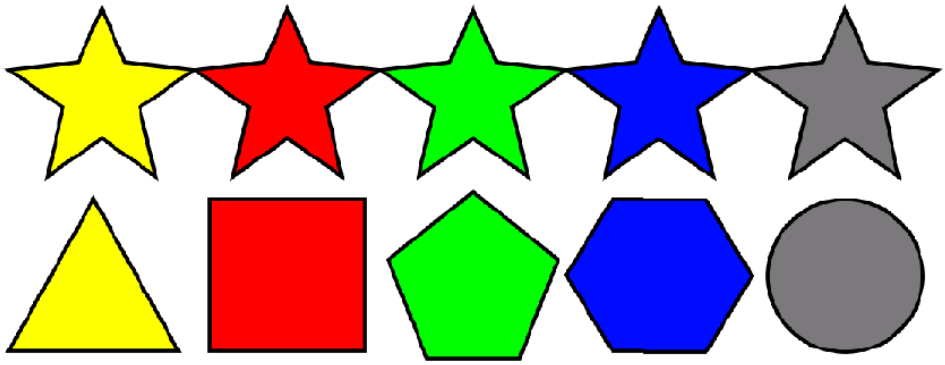


Figure 4.5: faMailiar message icons, most intimate on the left, least intimate on the right. The shapes on the top row are for outgoing messages, the bottom row for incoming messages. [Extracted from Mandic and Kerne [2005]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

monthly views. The daily view (see Figure 4.6) displays days along the x-axis and hours along the y-axis. A search box allows the filtering of messages by subject, body, sender and recipient. Additionally, subsets of messages can be highlighted in a hypertextual manner, e.g. displaying all messages sent by the same contact as the currently selected message or displaying only messages of the same thread. The weekly and monthly views are laid out similarly to the daily view, with the x-axis representing weeks and months and the y-axis days and weeks respectively.

4.6 Themail

Themail [Viégas et al., 2006] analyses the mailbox of a user to display relationships to other users. By looking at the context of messages exchanged between the owner of the mailbox and one contact, Themail extracts words that characterise the relationship over time. The main interface shows columns of extracted keywords, arranged along a timeline. Depending on frequency and distinctiveness over time, the keywords are drawn in different colours and sizes.

Multiple layers of information are visualised concurrently: yearly words display most used terms over an entire year and are shown as large, faint words in the background. Monthly words on the other hand represent frequently used words over the period of a month, arranged in columns along a timeline. The font size of those words is proportional to their distinctiveness and frequency. Additionally, keywords are selected to represent characteristical features of a specific relationship in contrast to the rest of the email archive.

Figure 4.7 shows the main interface of Themail. The yearly words in the background represent the most common words used in conversations over an entire year. Monthly words drawn in yellow in the foreground display a more detailed portrait of a person. Using a single column per month, they give a good sense of events that occurred during that time. Monthly words can be interactively selected (shown in white in Figure 4.7), to display a pop-up with all email messages containing that word. Individual email messages are drawn as circles, incoming and outgoing messages are distinguished by colour. The size of the circles is proportional to the length of the message.

Figure 4.8 shows the use of the search functionality which highlights matching keywords. The timeline at the bottom of the visualisation can be drawn either expanded or collapsed. Columns arranged along an expanded timeline are always placed at their real position, with months without any message simply left blank. Using a collapsed timeline, any spaces are omitted and only months containing messages are packed together.

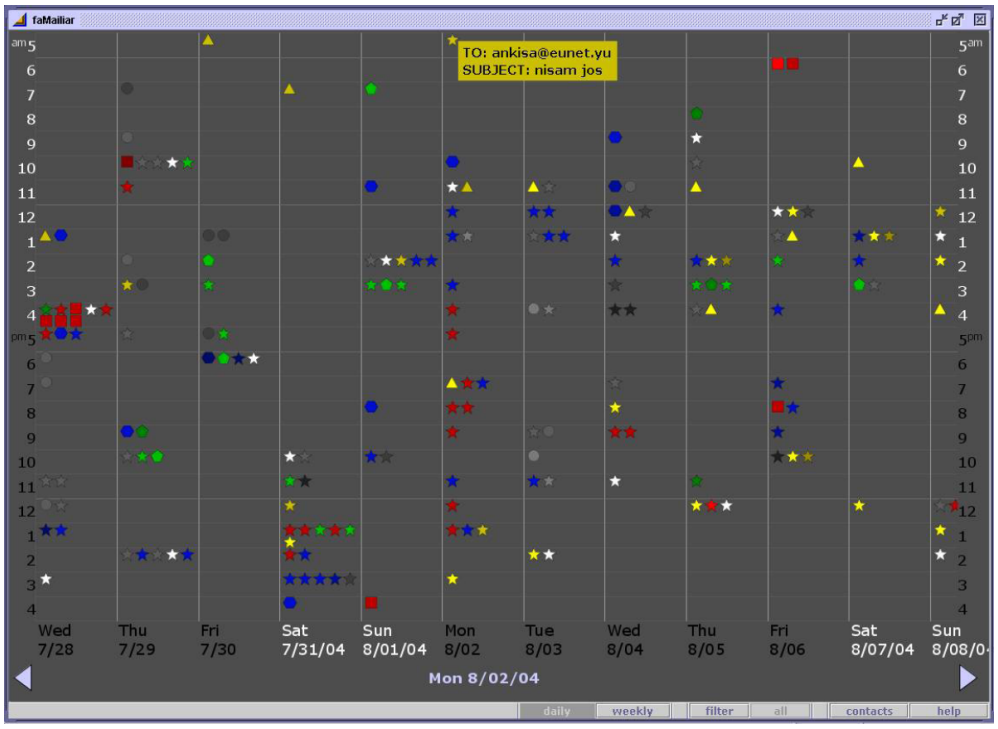


Figure 4.6: faMailiar daily view, showing all messages the user sent and received. Days are shown along the x-axis, hours of the day along the y-axis. [Extracted from Mandic and Kerne [2005]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

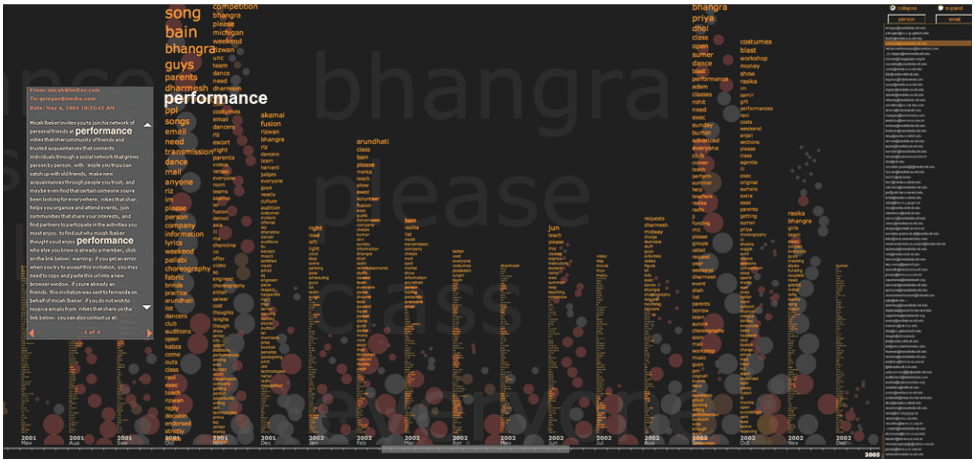


Figure 4.7: The main Themail interface. Yearly words in the background represent the most common words used in conversations over an entire year. Monthly words in the foreground display more detail. [Extracted from Viégas et al. [2006] under the terms of the ACM copyright. Copyright © by the Association for Computing Machinery, Inc.]

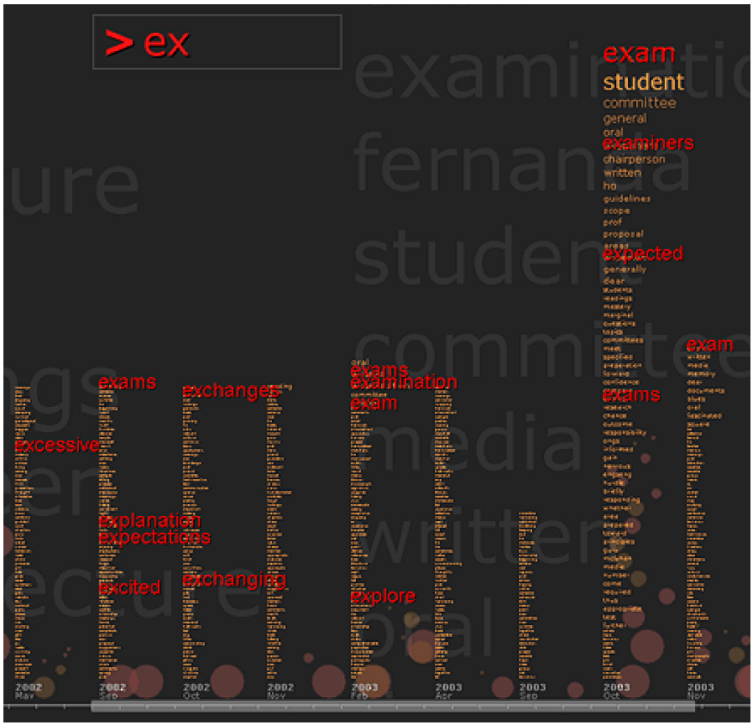


Figure 4.8: The Themail search interface with a timeline at the bottom. The user started to type a search term ("ex"), matches are highlighted. [Extracted from Viégas et al. [2006] under the terms of the ACM copyright. Copyright © by the Association for Computing Machinery, Inc.]

4.7 Email Conversations

Venolia and Neustaedter [2003] introduced a mixed model visualisation which displays an email message together with related messages to provide local context. Two models of conversation visualisation seem to exist: displaying messages as a simple sequence of turns (chronologically) and displaying the branching tree structure that is created by reply relationships.

To efficiently use the available screen space, conversations can be made the main unit of display. By shrinking a conversation to a single line, more conversations can be displayed at the same time. Additionally, operations that manipulate entire conversations can be provided. Deleting unwanted conversations could also delete any future messages of the same thread.

Venolia and Neustaedter [2003] propose a mixed model visualisation that lists all messages in a vertical list in chronological order, similar to a normal email client. Listing messages vertically makes scrolling necessary in one direction only. Replies to parent messages are indented in the chronological list to display the conversation tree structure. To provide an improved visualisation of parent-child relationships, messages are interconnected using lines running along the edges of the message boxes.

Figure 4.9 shows the main user interface of the proposed client. (2) shows a compact representation of the thread tree, constructed using a simple layout algorithm. Each message is assigned a unique row and column position. The rows are assigned in chronological order, the columns represent the tree structure of parent-reply relationships. Allowing the reuse of columns makes the visualisation extremely compact and keeps the width of the tree to a minimum.

The selected message is drawn in blue with its parent and child messages rendered in lighter blue colours. This helps the user focus on the corresponding branch of the message tree. The message boxes (1) normally show only a reduced set of information about a message (3) and can be expanded to show full message details. A summary of the entire conversation is displayed on the top (4), including the

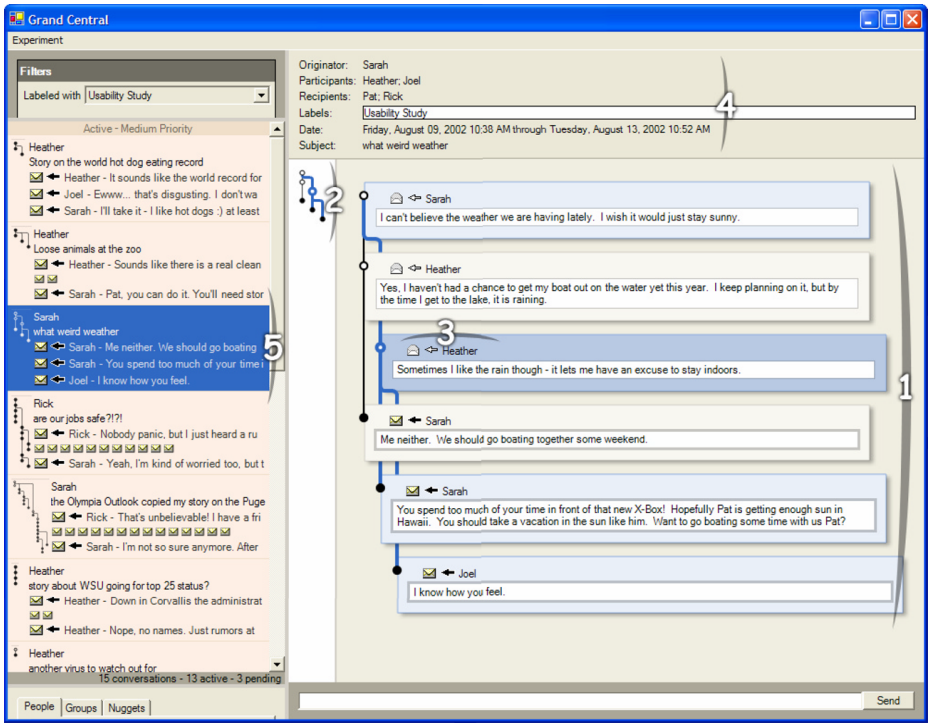


Figure 4.9: The proposed Thread Tree visualisation (2), a combined chronological and thread view, focusing on the conversation perspective. A summary of the conversation is displayed on the top (4). The conversation list (5) provides an overview over all conversations in the mailbox. [Extracted from Venolia and Neustaedter [2003] under the terms of the ACM copyright. Copyright © by the Association for Computing Machinery, Inc.]

originator of the thread, people who contributed to the thread (participants), and people only consuming the thread (recipients). A conversation list (5) on the left side provides an overview over all conversations in the user's mailbox. In this part of the visualisation, each conversation is reduced to a simple schematic, together with the name of the originator, the subject of the first message, and an indicator for unread messages.

A usability study with six participants was also conducted. A set of conversations was specifically generated for the study. At first, the test users were given five to ten minutes to explore the client interface. Afterwards, the users had to complete seven tasks with the client, such as finding a particular conversation or message and answering questions about the found message. In a post-test questionnaire, users rated the difficulty of the tasks and found the visualisation very easy to understand. Finally, the users were shown two paper screen shots of conversations and asked the same questions as before. Almost all participants were able to answer the questions in one or two seconds, with over 90% accuracy.

4.8 Conclusion

Different visualisation techniques and ideas covering email messages exist: from analysing the conversation structure, email archive search visualisations, the analysis of message contents using natural language processing, to visualising the social structures in email conversations. Most ideas provide separate visualisations, outside the context of normal mail clients. Such alternative views are rarely suited to be integrated directly into a mail client. Visualisations such as faMailiar or Themail can be interesting for the user to look at and explore archives, but they may fail to improve the day-to-day handling of email messages.

Other ideas such as ReMail and Email Conversations propose completely new user interfaces and try

to provide the user with an improved mail client. However, most users are accustomed to their current mail client and are reluctant to switch to a completely different program. This is where ideas like Thread Arcs (as part of ReMail, see also next chapter) excel, as they can easily be integrated into existing and widely adopted mail clients.

Chapter 5

Thread Arcs

> *"Thread Arcs provide a unique tool for interpreting email threads by elucidating the context of each message and by offering insight to the structure and evolution of email conversations."*
>
> [Kerr [2003a]]

Kerr [2003a,b] describes the Thread Arcs visualisation, which is part of the ReMail Prototype (see Section 4.1). It is an interactive visualisation technique which focuses on displaying threads found in email conversations. Thread Arcs visualise both the chronology of messages and the tree structure of a conversation in a mixed-model visualisation [Venolia and Neustaedter, 2003].

A thread is created by the "reply to" function in email clients, linking parent to child messages (which are replies to the parent). The first message which started a conversation is called root. The generational depth of a message is the distance (the number of replies) between the message and the root node.

Fisher and Moody [2001] state that threads found in email conversations are relatively small, 87% of all threads they analyzed had four messages or fewer. This places email threads in contrast to public discussion threads (for example on Usenet), which tend to grow large. Additionally, the "reply to" function in email clients is sometimes also used to start a new, unrelated conversation, which breaks the formal use of the reply function. This makes chronology an even more important attribute in email threads.

5.1 Qualities

Kerr [2003a] outlines seven essential qualities in visualising email threads:

Chronology: The sequence in which the messages of a thread arrive illustrates the growth of the thread over time. It also clearly shows which is the root message and which is the most recent message.

Relationships: The "reply to" relationships in a thread should be clearly visible, as they place a message in the context of the conversation.

Stability: Each message should stay in the same location as the thread grows, since this makes finding the same message in the future much easier.

Compactness: As screen real estate within an email client is limited, the visualisation should be small without affecting clarity.

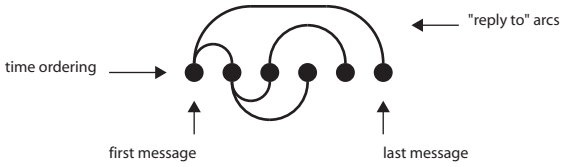
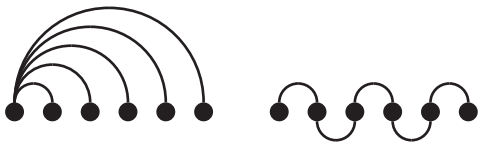


Figure 5.1: Relationships are shown with "reply to" arcs connecting nodes both above and below the message nodes. [Adapted from Kerr [2003a].]



(a) Bushy **(b)** Narrow

Figure 5.2: Bushy threads (a) have messages which receive two or more replies. Narrow threads (b) receive only one reply per message. [Adapted from Kerr [2003a].]

Attribute Highlighting: It should be possible to highlight messages with specific attributes, for example messages sent by a particular person or all unread messages.

Scale: The visualisation must handle both small and large threads found in email conversations. As threads grow in size, the clarity should degrade gracefully. As most conversations consist of between two and twenty messages [Fisher and Moody, 2001], it is not necessary to scale the conversation to hundreds of messages.

Interpretation/Sense: The type of conversation taking place in the visualised thread should be clearly visible, whether it is a back-and-forth reply between two people or multiple replies by a group of people to a single message.

5.2 Visualisation

The Thread Arcs visualisation consists of message nodes which are connected by relationship arcs. The nodes are laid out horizontally, the chronology is represented by the position of the node with the oldest message on the left and the most recent on the right. This makes the visualisation both compact and stable (see previous section). Relationship arcs are added connecting parent and child messages. To avoid overlapping arcs and make the visualisation easier to understand, the arcs are drawn both below and above the message nodes. The placement of an arc either above or below the message nodes has no particular significance. Figure 5.1 shows an example of a thread visualisation.

The visualisation immediately shows qualities such as the size of the thread (number of nodes) and the number of responses for a specific message (number of arcs leaving a node). Two distinct thread types can be differentiated: bushy threads have messages with two or more replies, narrow threads only have one reply per message. This difference is depicted in Figure 5.2.

The width of the visualisation is a linear function of the size (the number of messages) of the thread. The height is dictated by the number of replies (the number of stacked arcs) to a message. The visualisation can be made more compact by constraining the height of the arcs (see Figure 5.1). This way, Thread Arcs only grow horizontally and can be more easily integrated in the user interface of an email client. The algorithm to draw the Thread Arcs visualisation can be described by pseudo code shown in Listing 5.1 [Kerr, 2003a].

```
sort all messages chronologically
find the generation depth of each message

for each message
  if the message is the root message then
    place the node at the starting position
    do not draw an arc
  else
    place the message to the right of the last message
    if the message generation depth is odd then
      draw an arc above the line to the message's parent
    else
      draw an arc below the line to the message's parent
next message
```

Listing 5.1: Pseudo code for drawing Thread Arcs [from Kerr [2003a]]

Kerr [2003a] suggests to draw arcs alternately above and below the message nodes, depending on the generational depth of the message to avoid overlapping arcs. This ensures that incoming and outgoing arcs are always placed opposite of each other. More emphasis on existing sub-threads inside the conversation could be placed by keeping all arcs for a particular sub-thread either above or below the message nodes.

Figure 5.3 shows all possible Thread Arcs for thread sizes of two to five messages. Bushy (top left) and narrow (bottom right) threads can easily be told apart. As already discussed in Section 5.1, Thread Arcs is optimal for threads of around two to twenty messages. When displaying larger threads, Thread Arcs degrades gracefully. Although individual messages and details disappear, bushy and narrow structures within the total thread can be spotted. This helps the user obtain a feeling for the conversation taking place.

5.3 Existing Visualisation Techniques

Kerr [2003a] compares Thread Arcs (see Figure 5.4c) to Tree Diagrams (see Figure 5.4a) and Tree Tables (see Figure 5.4b), as a thread grows from one to six messages. In contrast to Thread Arcs, Tree Diagrams are not stable (the positions of message nodes change as new messages are added to the thread) and do not show the order in which the messages arrived. Rather, their emphasis is placed on the generational structure of a thread as each row in the diagram represents a new generation of messages. Considering that Tree Diagrams grow both wide (for bushy threads) and tall (for narrow threads), they are also not very compact. The same problems apply to Tree Tables.

Since both Tree Diagrams and Tree Tables do not display chronology, their visualisation of a thread is ambiguous. Figure 5.4 shows a conversation consisting of six messages. Thread Arcs (see Figure 5.4c) are able to show all eight possible ways the conversation may have taken place, whereas both Tree Diagrams (see Figure 5.4a) and Tree Tables (see Figure 5.4b) can only show a simplified version of the conversation since they lack information about chronology. To alleviate the missing chronology information, both Tree Diagrams [Smith and Fiore, 2001] and Tree Tables [Rohall et al., 2001; Venolia and Neustaedter, 2003] have been adapted to include information about the arrival sequence of email messages.

Thread Arcs look similar to Arc Diagrams [Wattenberg, 2002] (see also Section 2.1.4), although the two techniques vary in three important aspects. Arc Diagrams visualise repeating sequences in linear data such as strings, while Thread Arcs display the tree structure of email conversations. Thread Arcs

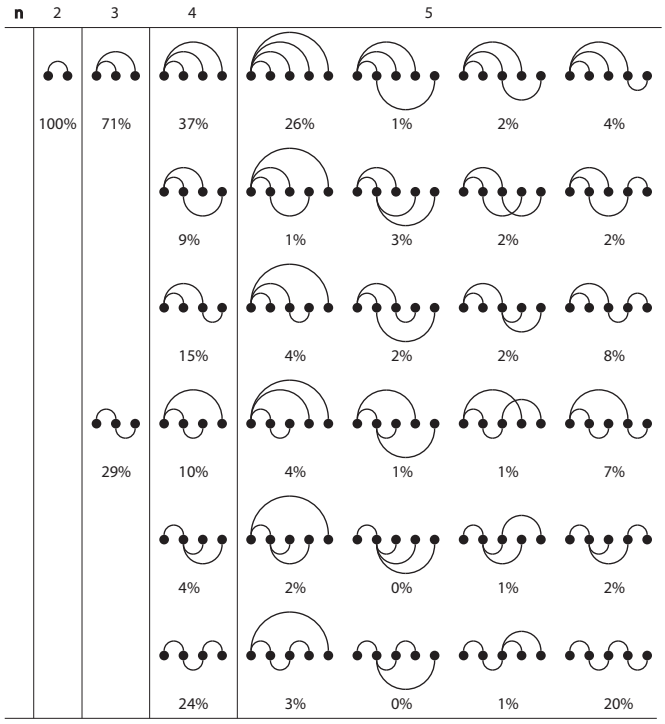


Figure 5.3: All possible Thread Arc visualisations for two to five messages. Percentages express the thread distribution in 42.000 analysed messages. [Redrawn from Kerr [2003b]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

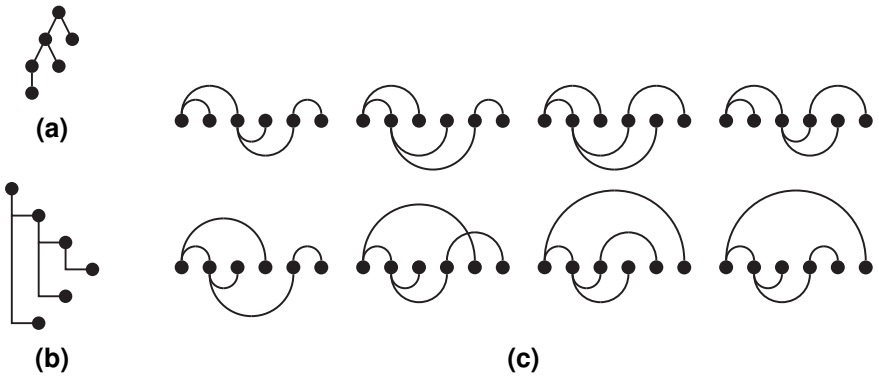
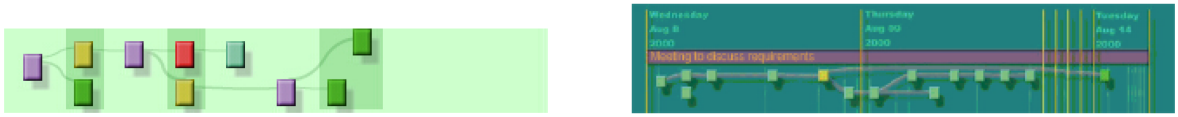


Figure 5.4: Tree Diagram (a) and Tree Table (b) oversimplify the conversational structure of a thread because they omit chronology. The eight Thread Arcs (c) show all the different ways the conversation may have evolved. [Redrawn from Kerr [2003a]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]



(a) (b)

Figure 5.5: Horizontal trees visualising email messages. (a) shows an email thread of ten messages. Colour indicates the relationship between the sender and the recipient. (b) shows a message tree with a timeline. Vertical lines display day boundaries. [Extracted from Rohall et al. [2001]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

connect single items (individual messages) with lines, while Arc Diagrams correlate repeating sequences (multiple items) with area. Arc Diagrams suffer from the same problem of overlapping arcs as Thread Arcs, but do not use the space below the line to improve understandability.

Figure 5.5 shows two examples of horizontal message trees presented in Rohall et al. [2001] and Fisher and Moody [2001]. Similar to Thread Arcs, messages are laid out horizontally in chronological order. In addition to showing the chronological order, a timeline also adds information about the time and date messages were sent. By using available vertical space to lay out the tree, emphasis is put on existing sub-threads in the conversation. Thread Arcs, in contrast, only use a single horizontal line to display messages of a thread.

5.4 Interaction

Thread Arcs are not a static visualisation, but also allow interaction by the user. The currently selected message in the mail client is highlighted with a blue hollow circle, all parent messages appear in a lighter blue colour, all child messages in a darker blue colour, unrelated messages fade out. Additionally, messages that fit certain criteria can be highlighted, such as all messages sent by a particular person, unread messages, or messages within a certain time frame.

For the ReMail prototype (see Section 4.1), two highlighting schemes have been chosen: people highlighting and attribute shading. Using hollow circles (instead of the normal "filled" circles), the people highlighting scheme emphasises the contributions to the conversation by a specific person. The attribute highlighting scheme allows the user to visualise the time the messages were received, the generational depth of each message or to assign a colour to each participant in the conversation. Schemes of highlighting are:

Personal highlights emphasises the contributions to the conversation by the user, which is a special case of the people highlighting scheme described previously.

Time shading uses the same colour for messages sent on the same day, giving the user a sense of the duration of the conversation.

Contribution colouring assigns a unique colour to each participant in the conversation.

Generational shading visualises the generational depth of each message using different shades. Since the messages are sorted chronologically, this information is less evident in the normal view.

5.5 Study

Kerr [2003a] also includes a study about the usefulness of the described thread visualisations. In total, the study conducted tests on eight participants and a total of 42.000 email messages. The users were

Figure 5.6: The prototype email client used in the usability tests conducted to evaluate the usability of Thread Arcs. The message visualisations (Thread Arcs, Tree Diagrams, Tree Tables) were included twice (A,B) using different highlighting schemes, users could switch between the different techniques during the test. [Extracted from Kerr [2003b]. Used under the terms of Austrian Copyright Law (UrhG) §46 [UrhG, 2008].]

introduced to message conversation visualisations and given examples of Thread Arcs, Tree Diagrams and Tree Table visualisations.

In the course of the test, users were asked to rank key qualities such as attribute highlighting, chronology, compactness, and stability they expect in a visualisation. Additionally, possible highlighting schemes such as contributor, generational, time, and unread were presented and ranked by the test users. The users were then asked to explore their own email archive using the presented email visualisations in a prototype of an email client built using Macromedia Director. Small tasks such as finding the last message of a conversation or finding the number of responses to a message encouraged the users to work with the different visualisation techniques. After experiencing the visualisations themselves, the users were asked to repeat the ranking exercise. At the end of the test, the users had to rate each visualisation against the key qualities presented earlier.

A statistical analysis showed that 38% of all messages were not part of a thread and that the most common size for an email conversation was three messages. Only a very small number of threads exceeded a size of 20 messages, with the biggest consisting of 483 emails. Of the 42.000 messages, 50% were part of threads that consisted of two messages or less, and 80% of all messages were part of threads of size five or less.

When asked to rank the presented key qualities after the test, users found relationships and chronology most important. The rating of the three presented visualisation techniques revealed that Thread Arcs fared best in displaying chronology, stability and compactness. Tree Diagrams ranked best in scalability, Tree Tables best in displaying relationships. Overall, Thread Arcs ranked best compared to the two other presented visualisations.

5.6 Conclusion

Kerr [2003a] concludes that Thread Arcs are a more stable and compact visualisation than Tree Diagrams and Tree Tables. It allows the user to see both the total number of messages in a thread, as well as the number of responses to a particular message. The linear, chronological layout helps the visualisation to stay compact and stable as new messages are added to existing threads.

Being a compact visualisation that is easy to understand without the need of extensive calculations makes Thread Arcs a prime candidate for integration into an existing mail client. It helps the user navigate through messages where normal mail clients fail: navigating a thread that spans across different folders, over an arbitrary timeframe. Most users keep their inbox sorted by date, which gets in the way of visualising thread connections. The visualisation restores this thread connection whilst allowing the inbox to continue to be sorted by date.

Chapter 6

Thunderbird Extensions

"... because X and C look similar if you beat them long and hard with a hammer."

[Matejka [2005]]

The basic design of all Mozilla applications is rather similar. A set of core features is implemented, compiled and distributed for different platforms and operating systems. Built upon this platform-dependent basis, platform-independent code for user interface and additional functionality is used. In the rest of the document, the term Mozilla is used to describe all applications that are built upon the Mozilla application and share the same features, such as the Mozilla Thunderbird mail client, the Mozilla Firefox web browser, or the Mozilla Sunbird calendar application.

Mozilla applications support extending the standard application by so-called "add-ons": Themes (or skins) alter the appearance of the application. Plugins enable the application to handle content it can not handle natively, such as PDF files or movies. Extensions add additional functionality to the application, such as removing unwanted advertisements from web pages or adding a weather forecast icon.

Developing extensions for Mozilla is fairly simple. An extension consists of files describing the user interface and application code written in JavaScript. Additionally, more complex operations can be implemented using C or C++.

The user interface description uses a special XML dialect, the XML User Interface Language (XUL). XUL supports a wide range of built-in widgets (see Section 6.1), even more widgets can be constructed using the XML Binding Language (XBL) (see Section 6.2). Extensions can alter the user interface of the application without changes to the original source code using overlays (see Section 6.1.5).

JavaScript is used to connect those widgets and implement the functionality of the extension. To perform more complex operations, parts of the extension can be implemented in C or C++. To access methods provided by these modules, the Cross Platform Component Object Model (XPCOM) (see Section 6.4) and the XPConnect technology (see Section 6.5) enable the integration of external native libraries in JavaScript. XPConnect is also used to provide developers access to functionality of the application like sending or retrieving mail messages or accessing the address book.

All files that make up an extension are packed together and form a Cross Platform Installer Module (XPI), which can be easily distributed, downloaded, and installed (see Section 6.7). Mozilla provides for package management, including automatic updates to installed extensions and easy deinstallation.

Currently, literally hundreds of different extensions for Firefox and Thunderbird exist. The functionality ranges from simple user interface changes like adding a close button to tabs in the browser to complex extensions that remove unwanted advertisements from websites.

6.1 XML User Interface Language (XUL)

The XML User Interface Language (XUL) (see Mozilla [2006j]) is based on the Extensible Markup Language (XML) and was created to make the development of the Mozilla browser easier and faster. Mozilla [2006k] gives an overview of XUL and describes its use within the Mozilla project.

Separating application code and user interface description helps build portable user interfaces, which can be easily implemented and modified. Although XUL aims to be platform-independent, some platform-specific work by the user interface designers might be needed (for example the placing of OK and CANCEL buttons in dialogue boxes). The resulting XUL code can only be platform-independent to a certain degree and separate, platform-specific versions of some interface components will have to be maintained.

XUL supports all kinds of different user interface widgets like text boxes, check boxes, toolbars, menus, pop-up menus, tabbed dialogues, etc. The interface components displayed on screen can be created by the contents of the XUL file itself (buttons, descriptions) or by loading data from external data sources (menus, mailboxes, bookmarks, search results).

Internally, the rendering of XUL is handled by the same rendering engine that renders HTML content. So, naturally, the process in which the abstract definition of the user interface is transformed into displayable content resembles the process of displaying HTML pages. The rendering engine converts the content into a set of objects which can be displayed on screen. These objects are organised as a document tree similar to the Document Object Model (DOM) used in displaying HTML content. Just as in HTML, Cascading Style Sheets (CSS) are used to control the layout of the user interface. XUL supports almost all "normal" CSS properties and adds some properties specific to Mozilla and XUL.

As the whole user interface of Mozilla is designed using XUL files, these files need to be managed and put into a hierarchy of some sort. To distinguish between files that are part of the user interface and external files, Mozilla uses a special URL to access user interface files, the `chrome://` URL. Files accessed using this URL are able to perform privileged operations such as accessing local files, preferences, or bookmarks.

6.1.1 XUL File Structure

XUL uses a fairly simple file structure, an example is given in Listing 6.1. In line 1, the file is identified as an XML file, line 2 includes a reference to an external stylesheet, in this case to a directory containing style sheets. Line 4 defines a new `window` element, lines 5 to 7 define attributes for this element. In line 8, the default namespace for all following elements is defined as the XUL namespace [1]. Line 10 closes the `window` element. The actual content of the window is put between the opening and closing `window` tag.

6.1.2 XUL Elements

As previously mentioned, XUL supports a wide range of different user interface widgets, some of which shall be presented here. A more complete overview can be found in Mozilla [2006k], Mozilla [2001], and XULPlanet [2006].

Buttons

The most basic tag to create a user interface element is the `button` tag. A very simple example is shown below:

[1] Mozilla [2006k] notes that this URL is never actually downloaded but is recognised internally. If you open the URL in a browser that understands XUL, it says "There is no data, there is only XUL.".

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<window
    id="findfile-window"
    title="Find Files"
    orient="horizontal"
    xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
    >
<!-- Widgets and markup go here -->
</window>
```

Listing 6.1: An example XUL file (taken from Mozilla [2006k]).

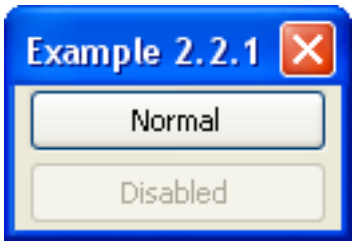


Figure 6.1: Two XUL buttons. [Taken from Mozilla [2006k].]

```
<button id="id1" label="Normal"/>
<button id="id2" label="Disabled" disabled="true"/>
```

`id` defines a unique identifier used to identify the button within code or a style sheet (cf. HTML, DOM). The `label` is the text that appears on the button. Figure 6.1 shows how this code is rendered by Mozilla (when placed inside a window, see Listing 6.1).

Labels and Descriptions

`Labels` and `descriptions` are handled internally exactly the same way. By convention, a `label` is used as a label for a control, whereas a `description` should be used for other descriptive text. The text to be displayed can be defined by including a `value` property in the tag (displayed as a single line of text) or by placing it between the opening and closing tag (wrapped onto multiple lines if necessary). The example below shows the use of a `label` and a `description` tag, Figure 6.2 shows how both these elements are rendered:

```
<label value="This is a single line of text." />
<description>
This is a longer text that will automatically wrap onto multiple lines if
    necessary.
</description>
```

Images

Images can be included in XUL files using the `image` element. The URL is specified using the `src` attribute. An example is given below:

```
<image src="path/to/file.jpg" />
```

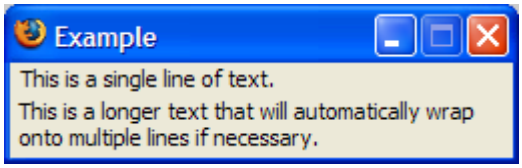


Figure 6.2: XUL text elements.

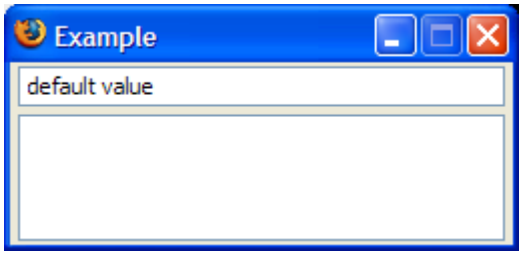


Figure 6.3: A text entry field and a text entry box in XUL.

Text Entry Fields

Text entry is handled by using the `textbox` element. This creates a box in which the user can enter text. By specifying the `multiline` attribute, a larger box can be displayed (c.f. `textarea` in HTML). A default text can be supplied by the `value` attribute. The code below shows two `textbox` elements, Figure 6.3 shows how these elements are displayed.

```
<textbox value="default value" />
<textbox multiline="true" rows="3" cols="40" />
```

Check Boxes

A `checkbox` element can be switched between two states by the user: checked and unchecked. The `checked` attribute sets the default state, a `label` attribute can be used to display a descriptive text beside the check box.

Radio Buttons

To give the user more choices than on or off, `radio` buttons can be used. These elements are normally grouped together using a `radiogroup` element. In such a group, only one radio button can be selected at the same time. Listing 6.2 shows a simple `checkbox` and three `radio` buttons grouped together, their appearance in the user interface is illustrated in Figure 6.4.

6.1.3 XUL Box Model

In XUL, a window is divided into a series of boxes, which themselves contain other boxes. This so-called box model is the main form of layout in XUL. The combination of boxes, spacers, elements with flex attributes and CSS style properties control the layout of the window.

The children of a `box` can be laid out in two orientations, either horizontally or vertically. This can be achieved by setting the `orient` attribute or by using two specialised elements, the `hbox` and `vbox`. Listing 6.3 shows two simple boxes, the first oriented horizontally, the second vertically. Figure 6.5 displays the rendering of those two boxes.

```
<checkbox checked="true" label="Some descriptive text" />
<radiogroup>
  <radio label="One" />
  <radio label="Two" />
  <radio label="Three" />
</radiogroup>
```

Listing 6.2: Check boxes and radio buttons in XUL.

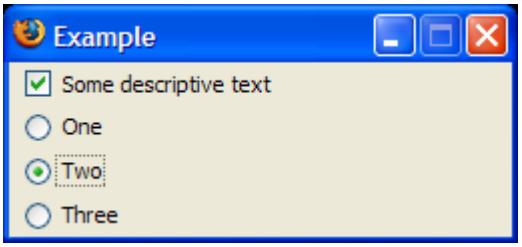


Figure 6.4: Check boxes and radio buttons in XUL.

```
<hbox>
  <button label="These" />
  <button label="buttons" />
  <button label="are" />
  <button label="oriented" />
  <button label="horizontally." />
</hbox>

<vbox>
  <button label="These" />
  <button label="buttons" />
  <button label="are" />
  <button label="oriented" />
  <button label="vertically." />
</vbox>
```

Listing 6.3: Two simple boxes in XUL, the first arranged horizontally, the second vertically.

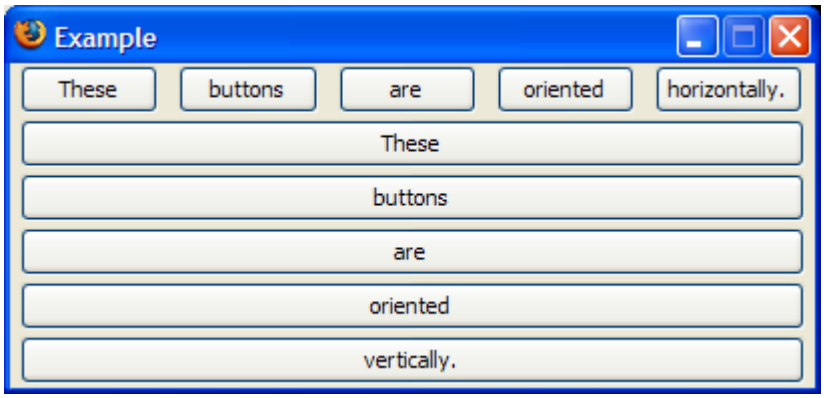


Figure 6.5: Two simple boxes in XUL, corresponding to the code in Listing 6.3.

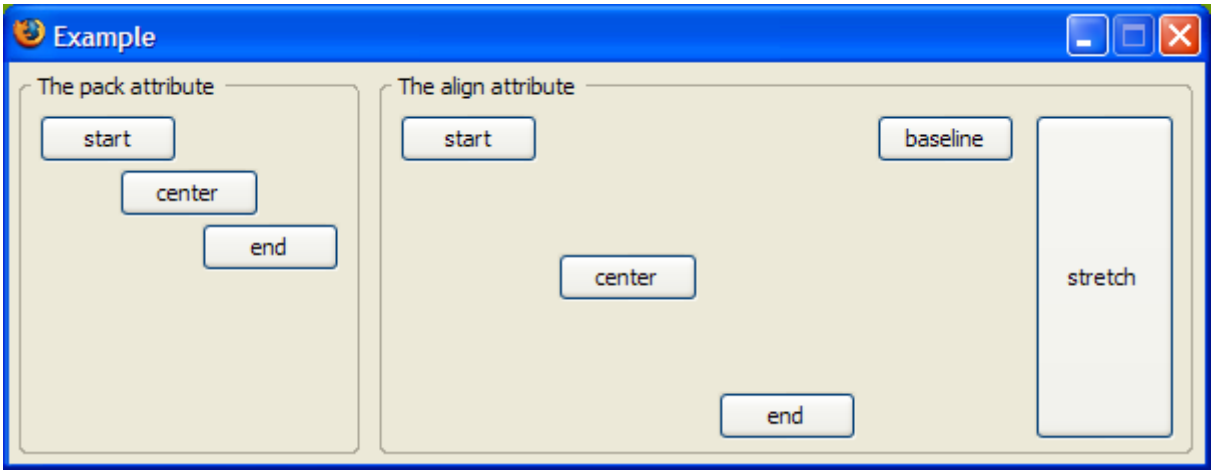


Figure 6.6: Various positioning options for a horizontal XUL box. For a horizontal box, the pack attribute works horizontally and the align attribute vertically.

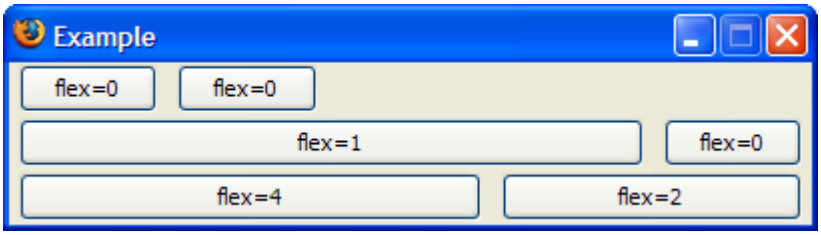


Figure 6.7: Using the `flex` attribute to control layout in XUL.

Additionally, a specific width or height can be set on each child element or a minimum and maximum size can be given. The layout of child elements inside the box is further defined by the `pack` attribute. It controls the horizontal positioning of the children for horizontal boxes, the vertical positioning for vertical boxes. Possible values are `start` (left/top), `center` (centred) and `end` (right/bottom).

To control the positioning along the second axis, the `align` attribute is used. It controls the vertical position of child elements for horizontal boxes and the horizontal position for vertical boxes. Possible values are `start` (top/left), `center` (centred), `end` (bottom/right), `baseline` (to line up to text, only useful for horizontal boxes) and `stretch` (default, causes elements to grow to fit the parent box). Figure 6.6 shows all possible positioning options for a horizontal box.

6.1.4 Layout

The Flex Attribute

Strictly speaking, the `flex` attribute is not a layout element. It defines how available screen space is divided among the user interface elements. The higher the `flex` attribute, the more space is given to the element. Figure 6.7 shows the same two buttons with different flex attributes. In the first row, no `flex` attribute is set on both buttons. Thus both are rendered with their default width, the rest of the available space is not used. The second row shows what happens when the `flex` attribute is set for only one element: it receives all the remaining space. In the third row, a `flex` attribute is set for both elements. Here, the available space is divided proportionally between the elements. The higher the `flex` attribute, the more space is assigned to the element.

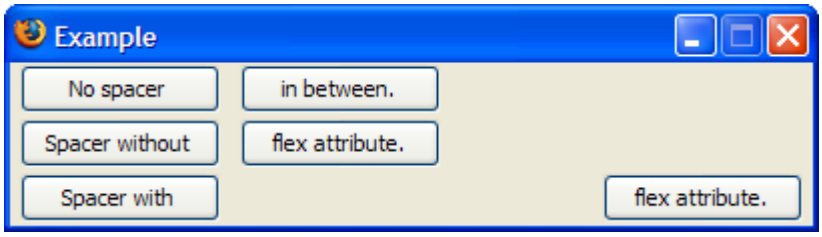


Figure 6.8: Using the `spacer` element to add spacing between elements.

```
<groupbox>
  <caption label="Caption" />
  <button label="Button" />
  <label value="Label" />
</groupbox>
```

Listing 6.4: An XUL groupbox.

Spacers

Spacers are layout elements which simply create blank space between other elements. The only useful attribute for a `spacer` element is the `flex` attribute. Figure 6.8 shows the use of a single `spacer` element between two buttons. The first row shows the two buttons without any spacer in between. In the second row, a `spacer` element is added, but no `flex` attribute is set. The `spacer` element in the third row has its `flex` attribute set, so it takes up all available space between the two buttons.

Groupboxes

A `groupbox` is used to group similar elements together. This is reflected in the user interface by a bevelled border that is drawn around the grouped elements. Additionally, a caption can be placed along the top part of the border. Listing 6.4 shows the code used to create the group box presented in Figure 6.9.

Stacks

A `stack` element works like any other box, except that its child elements are laid out on top of each other instead of being placed side-by-side. The position of the child elements is either determined by specifying `left` and `top` attributes or using CSS style rules. When the child elements are drawn, the order in which they were added to the `stack` becomes important. The rendering engine simply renders all elements in the order they appear inside the `stack` element, meaning that the first element in the `stack` appears on the bottom, the last element on the top. Listing 6.5 shows a simple `stack` with three buttons, Figure 6.10 shows how the elements are rendered on screen.

6.1.5 Overlays

The more complex an application becomes, the more common elements such as menus it will contain. Instead of re-implementing these elements in every window, a mechanism to share such common elements is needed. This is where `overlays` come into play. An `overlay` is an XUL file which describes extra content to be added to an existing user interface. During the rendering process, the base XUL file and all included overlays are combined to form the final code base for the user interface. [Mozilla, 2005]

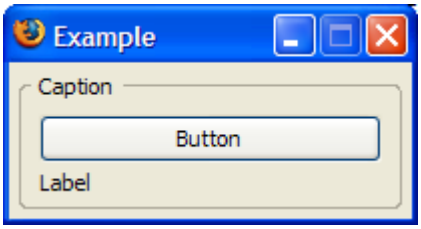


Figure 6.9: An XUL groupbox.

```
1 <stack>
2   <button left="10" top="10" label="Button 1" />
3   <button left="40" top="30" label="Button 2" />
4   <button left="30" top="50" label="Button 3" />
5 </stack>
```

Listing 6.5: An XUL `stack` with three buttons.

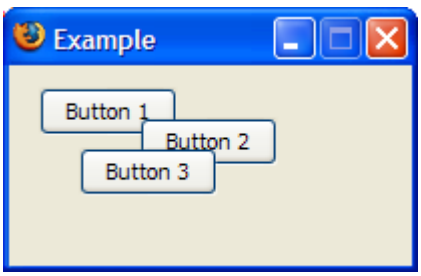


Figure 6.10: An XUL `stack` with three buttons.

```
<?xml version="1.0"?>

<overlay id="overlay"
    xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<groupbox id="box1">
  <button label="This button was added using an overlay." />
</groupbox>

<groupbox id="box2">
  <description>And even more descriptive text.</description>
</groupbox>

</overlay>
```

Listing 6.6: A simple XUL overlay.

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<?xul-overlay href="chrome://path/to/overlay.xul"?>

<window
   xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
   title="Example">

<groupbox id="box1">
  <button label="This button is defined in the base file." />
</groupbox>

<groupbox id="box2">
  <description>Some description.</description>
</groupbox>

</window>
```

Listing 6.7: The base XUL file for the `overlay` defined in Listing 6.6.

This mechanism can not only be used to avoid duplicate code, but also to provide a simple means to extend the user interface by custom code. All extensions for Mozilla which need to display elements on screen use `overlays` to include their code without modifying the original user interface code. An example for a simple `overlay` is given in Listing 6.6, the base XUL file to be combined with this `overlay` in Listing 6.7.

Inside the top `overlay` element, one or more child elements can be defined. Note that those elements are also defined in the base XUL file. During rendering, the layout engine uses the `id` attribute to match the elements from the overlay to the base file. In this case, it matches the two `groupboxes` in Listings 6.6 and 6.7. All elements defined in the `overlay` are added to the elements in the base file. Basically, the two files are merged together and the final output is rendered. The left window in Figure 6.11 shows the base XUL file without including the `overlay`, the right window shows the combination of both.

Overlays can be loaded explicitly or dynamically. In the previous example (see Listing 6.7) the `overlay` was included explicitly using an XML processing instruction. Using explicit loading, the `overlay` is loaded every time the base XUL file is displayed. This is useful to share common elements

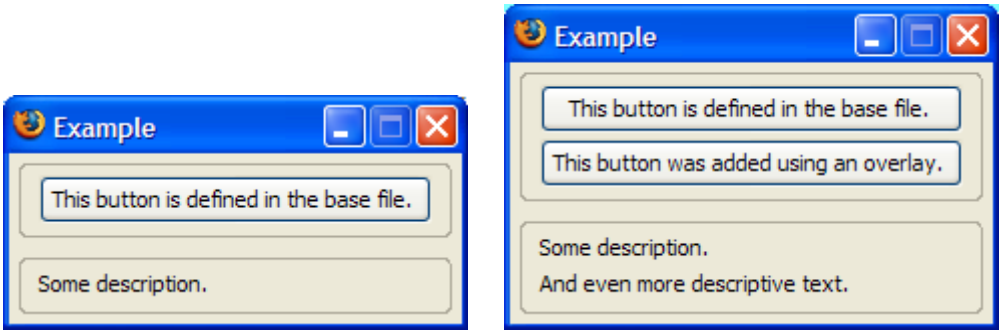


Figure 6.11: XUL `overlays`: the base XUL file on the left, the final rendering on the right.

such as menus. An example of such a processing instruction is shown below:

```
<?xul-overlay href="chrome://path/to/overlay.xul"?>
```

Due to the fact that the base XUL file loads the `overlay` explicitly, a change to the base file is needed to include another `overlay`. This makes this approach unusable for extension development, as a change to the original code is needed. Using the chrome registry (see Section 6.3.2), dynamic overlays are possible. In this case, the overlay is not defined in the base XUL file, but is separately managed in the chrome registry and applied at run-time. So, instead of defining the `overlay` in the base XUL file as shown above, it is defined in a manifest file as can be seen below:

```
overlay chrome://path/to/basedocument.xul chrome://path/to/overlay.xul
```

6.2 XML Binding Language (XBL)

The XML Binding Language (XBL)[2] (see Mozilla [2006f]) is a language which enables developers to attach new behaviour to existing elements. A binding can extend an element by adding new event handlers, by implementing additional methods and properties, or by inserting new content. XBL is used to implement most XUL widgets. The XBL 1.0 Specification can be found in Mozilla [2006g], but as Mozilla [2006f] notes "... the actual implementation in Mozilla is different from the specification, and there's no known document describing the differences.".

Mozilla [2006m] describes how XBL can be used in Mozilla application development. An XBL file may contain one or more `bindings`, and each `binding` describes the behaviour to be added to an existing element. As XBL is an XML language, the style is fairly simple. Listing 6.8 shows the skeleton of an XBL file. In the file, two bindings are defined and can be referenced by their `ids`.

The `bindings` element is the top element and may contain one or more `binding` elements that each declare a single binding. The bindings that are declared in the XBL file may be attached to any XUL element. To link the XBL `binding` to an existing element, a special CSS property `-moz-binding` is used. Listing 6.9 shows how any element of class `scrollbar` is bound to `binding1` defined in the XBL file from Listing 6.8. More information about XBL and its possibilities can be found in Mozilla [2006m].

6.3 Chrome

Chrome is the name for all user interface components except the main content area. This includes toolbars, menu bars, progress bars, and similar elements. Mozilla [2008a] describes the use of chrome in

[2] Sometimes also called Extensible Bindings Language.

```
<?xml version="1.0"?>
<bindings xmlns="http://www.mozilla.org/xbl">
  <binding id="binding1">
    <!-- content, property, method and event descriptions go here -->
  </binding>
  <binding id="binding2">
    <!-- content, property, method and event descriptions go here -->
  </binding>
</bindings>
```

Listing 6.8: Basic skeleton of an XBL file (taken from Mozilla [2006m]).

```
scrollbar {
    -moz-binding: url('chrome://path/to/file.xml#binding1');
}
```

Listing 6.9: Example XBL binding using CSS (taken from Mozilla [2006m]).

the Mozilla project. To access parts of the chrome, a special `chrome://` URL is used. By using this special URL, files receive elevated privileges such as reading and writing to local files, access preferences, or other privileged operations.

6.3.1 Chrome Providers

A chrome provider is a supplier of chrome for a given window type. Three basic types of chrome providers exist:

content: The content provider supplies the main source file (typically an XUL file) for a window. JavaScript files belonging to the user interface and XBL binding files are also part of the content package.

locale: The locale provider stores all localised information for an application. By using a different provider for content and localisation, translators can easily translate an application without changing the rest of the source code.

skin: The skin provider supplies all files that describe the visual appearance of the application, typically CSS files and images.

Although XUL aims to be platform-independent, chrome packages can be marked as platform specific, containing different sets of files for different platforms.

6.3.2 Chrome Registry

The chrome registry provides mappings from chrome package names (the `chrome://` address space) to the physical location of those files on disk. The registry is configurable and persistent and allows the user to select a preferred skin or locale. Available chrome needs to be added to the chrome registry using manifest files. In earlier versions of Mozilla, an RDF format was used, whereas newer versions use a simple plain text format.

```
1 content packagename path/to/files
2 locale packagename localename path/to/files
3 skin packagename skinname path/to/files
4 overlay chrome://file-to-overlay chrome://overlay-file
5 style chrome://file-to-style chrome://stylesheet-file
```

Listing 6.10: An example plain text chrome manifest.

6.3.3 Chrome Manifests (Plain Text)

Plain text chrome manifests use a simple line-based format. Listing 6.10 shows a basic manifest file. In line 1, a new content package is registered. This maps the path `chrome://packagename/` to the specified local path. Line 2 adds a new locale (`localename`, for example `en-US`) for the package. Line 3 adds a new skin (`skinname`) for the package. Line 4 defines an overlay, that is `chrome://overlay-file` overrides the file `chrome://file-to-overlay`. Finally, line 5 adds the stylesheet `chrome://stylesheet-file` to the XUL file `chrome://file-to-style`.

6.4 Cross Platform Component Object Model (XPCOM)

The idea behind the Cross Platform Component Object Model (XPCOM) (see Mozilla [2000] and Parrish [2001b]) is to provide a modular, cross-platform, and language-neutral framework for writing software. Applications built on XPCOM use a set of core libraries to load and manipulate XPCOM components. These components can be implemented in C, C++, or JavaScript (see Section 6.5) and can in turn be used from C, C++, and JavaScript. XPCOM supports most platforms which host a C++ compiler, most notably Windows, Linux, and MacOS. Parrish [2001a] gives a more detailed description of XPCOM and its use.

By using a common format for describing methods, attributes, parameters, and interfaces of XPCOM components, the same interfaces can be described independently of platform and programming language. This common format is provided by the type library. A special Interface Description Language (IDL) is used to provide a language-neutral way to specify interfaces of XPCOM components. Using an IDL compiler, a type library file is created from the interface description. The Mozilla IDL compiler - the XPIDL compiler - has the additional feature of creating C++ code stubs from the language-neutral interface description. This helps tremendously in getting started a on new project.

Listing 6.11 shows a simple interface using the XPIDL syntax (see Mozilla [1998]). The `interface` keyword is followed by the name of the interface, a colon and the name of the base interface. A pair of opening and closing curly braces form a block containing the definition of all methods and attributes. All attribute definitions are prefixed with an `attribute` keyword, the `in`, `out` and `inout` prefixes are used to declare parameters to methods as input, output, or input and output parameters.

The interface in Listing 6.11 is named `nsIScreen` and inherits from `nsISupports`. Two methods, `GetRect` and `GetAvailRect`, and two attributes, `pixelDepth` and `colorDepth`, are defined. The `readonly` keyword allows both attributes to be read but not set. In line 2, optional metadata is added to the interface description. Using the `scriptable` keyword, the interface can be used from scripting languages such as JavaScript. The `uuid` keyword defines the interface's Universally Unique Identifier (UUID) (see ISO/IEC [2004]) which is used in interface discovery described below.

```
#include "nsISupports.idl"
[scriptable, uuid(f728830e-1dd1-11b2-9598-fb9f414f2465)]

interface nsIScreen : nsISupports
{
  void GetRect(out long left, out long top, out long width, out long
     height);
  void GetAvailRect(out long left, out long top, out long width, out long
      height);
  readonly attribute long pixelDepth;
  readonly attribute long colorDepth;
};
```

Listing 6.11: Example XPIDL interface description (taken from Parrish [2001a]).

```
interface nsISupports
{
    void QueryInterface(in nsIIDRef uuid, out nsQIResult result);
    nsrefcnt AddRef();
    nsrefcnt Release();
};
```

Listing 6.12: `nsISupports` interface definition (taken from Parrish [2001a]).

6.4.1 Interface Discovery

As already described, XPCOM uses a strict interface-based approach to handling components. Client code can only communicate with components through their interfaces. The interface `nsISupports`, from which any interface is derived, supports a mechanism to determine what interfaces are supported by the specific component, and if it supports more than one interface, to switch between those interfaces. A simplified interface definition of the `nsISupports` interface can be found in Listing 6.12.

Using the method `QueryInterface`, interfaces implemented by the component can be discovered. The parameter `nsIIDRef` specifies the UUID of the interface which should be used. If this interface is not supported by the component, an error code is returned as `nsQIResult`, otherwise the address of the interface and a success result code are returned. Listing 6.13 shows the use of the `QueryInterface` method to specify which interface should be used to interact with the component `file`.

The string `@mozilla.org/file/local;1` that is used in this example is called a contract id. Components can be created using the component manager by either specifying their class id (a UUID) or their contract id (a simple text string). Parrish [2001a] recommends to use the format `@<internetdomain>/module[/submodule[...]];version[?<name>=<value>[&<name>=<value>[...]]]`. The `version` parameter in the above example is 1. It is used to distinguish between different versions of the same component, but does not imply backwards compatibility.

6.4.2 Lifetime Management

The lifetime management of XPCOM components uses a simple reference counting mechanism. Using the `AddRef` method of the `nsISupports` interface, the internal reference count can be incremented. Calling the `Release` method decrements the reference count. Components must keep track of any interfaces that have been issued to avoid destroying the component before all client code is completed. If a `QueryInterface` call is successful, it implicitly calls `AddRef` on the component to register the

```
1 // first, we create an instance of something...
2 var file = Components.classes["@mozilla.org/file/local;1"].createInstance
      ();
3 // second, we specify which interface we actually want to use.
4 file = file.QueryInterface(Components.interfaces.nsIFile);
5 // do something generic with the nsIFile interface here.
6 file.create(NORMAL_FILE_TYPE, 0377);
7 var size = file.fileSize;
```

Listing 6.13: Using XPCOM interfaces in JavaScript (taken from Parrish [2001a]).

use of the issued interface. Client code must then explicitly call `Release` on any interface it is done using. When the internal reference count reaches zero, the object is destroyed. A more detailed view of XPCOM, component internals, and component creation, and an API reference can be found in Turner and Oeschger [2003].

6.5 Scriptable Components (XPConnect)

As explained in Section 6.4, XPCOM components can be written in and accessed from C and C++. Using XPConnect, it is possible to transparently access XPCOM components from JavaScript and implement XPCOM objects using JavaScript (see Bandhauer [2000]). The Document Object Model (DOM) is an example where access to JavaScript objects from native code is necessary.

The XPIDL compiler (see Section 6.4) creates the type library file, which is used by XPConnect to build proxy objects (wrappers) between JavaScript and C++ code. XPCOM objects implemented in C++ are wrapped in a `JS2CPPProxy` object, which emulates a native JavaScript object and forwards calls to the underlying XPCOM component. A `CPP2JSProxy` object provides access to JavaScript objects from C++.

As the `JS2CPPProxy` object is a native JavaScript object, it is subject to the garbage collection in the JavaScript runtime. Any references held by the object are released upon finalisation. This simplifies the handling XPCOM components from JavaScript code.

The XPConnect `Components` object exposes XPConnect functionality in JavaScript. It is itself a wrapper for the `nsIXPCComponents` interface, which is reflected into JavaScript using XPConnect. Two important members of the `Components` object (which have already been used in Listing 6.13) are described below, the whole reference can be found in Mozilla [2008d].

Components.classes

The `Components.classes` object is a read-only array containing all classes of XPCOM objects that can be constructed or accessed as a service. The elements in the array are indexed by their contract id (see Section 6.4.1). Listing 6.14 shows how to obtain a reference to an XPCOM class using the contract id and create an instance or access a service based on this class. The `getService` and `createInstance` methods return an XPConnect wrapped XPCOM object that only supports one interface, the `nsISupports` base interface.

Components.interfaces

The `Components.interfaces` object is a read-only array containing all interfaces for the available XPCOM objects. Since an interface must be explicitly marked as `scriptable` in its XPIDL definition,

```
1 var clazz = Components.classes['@mozilla.org/preferences;1'];
2 var obj = clazz.getService();
3 // or
4 var clazz = Components.classes['@mozilla.org/messenger;1'];
5 var obj = clazz.createInstance();
```

Listing 6.14: Creating an XPCOM object in JavaScript.

```
1 var iface = Components.interfaces.nsIPref;
2 var srv = obj.QueryInterface(iface);
```

Listing 6.15: Querying an interface of an XPCOM object in JavaScript.

the array contains only those interfaces. The elements in the array are indexed by their interface name.

Using the `obj` object from Listing 6.14 which contains a reference to the `nsISupports` base interface of the XPCOM object, a specific interface can be requested. The `iface` object stores a reference to this interface which is retrieved from the `Components.interfaces` array. Calling the `QueryInterface` method on the `nsISupports` interface returns a reference to the desired interface for the XPCOM object.

In summary, the access to an XPCOM object from JavaScript requires the following steps (see Listings 6.14 and 6.15):

- Acquire a reference to the class of the object through the `Components.classes` array (`clazz`).
- Create an instance of this class by calling `createInstance` (`obj`).
- Define the XPCOM interface to access the object through the `Components.interfaces` array (`iface`).
- Acquire a reference to this interface on the object by calling the `QueryInterface` method (`srv`).

6.6 JavaScript

All interactions between user interface elements and extensions in Mozilla Thunderbird are implemented in JavaScript. Being an interpreted language, extensions written in JavaScript can be installed on any platform and operating system. The disadvantage of the JavaScript interpreter is that only a single thread executes both the user interface of the mail client and all the code of any installed extensions. This drawback has to be kept in mind when extensions need to perform long-running, processor intensive work, since this will disrupt the normal behaviour of the mail client.

JavaScript is a dynamic scripting language. It greatly differs from class-based languages such as Java or C++, since it is a prototype-based language. In class-based languages, classes and instances are two distinct entities. The abstract class defines all properties of the entity, an instance is an instantiation of the class. It has all the properties of the class, no more and no less.

In prototype-based languages, there is no distinction between classes and instances. A prototypical object is used as a template from which the initial properties of a new object are retrieved. In contrast to class-based languages, any object can specify its own, additional properties. As such, a JavaScript object is simply a collection of name-value pairs, similar to a `Map` in Java or an associative array in PHP.

```
1  function Counter(startValue) {
2      this.value = startValue;
3  }
4
5  Counter.prototype.increase = function() {
6      this.value++;
7  }
8
9  Counter.prototype.decrease = function() {
10     this.value--;
11 }
```

Listing 6.16: A simple JavaScript counter.

JavaScript can be used both as a procedural and object-oriented language. Objects are created at run-time by attaching methods and properties to empty objects. As mentioned before, this is in contrast to class-based languages, where all properties and methods are defined at compile time in the class definition. There are several primitive types supported by the JavaScript runtime (see Mozilla [2008c]):

- Number
- String
- Boolean
- Object
  - Function
  - Array
  - Date
  - RegExp
- Null
- Undefined

Listing 6.16 shows the object definition for a simple counter with an instance variable and two methods. Calling the `increase` method increases the counter, calling `decrease` decreases the counter.

6.7 Package Management

An extension is typically packaged as Cross-Platform Installer Module (XPI) file (see Mozilla [2006h]), which is a ZIP file containing all files comprising the extension. These extensions can be downloaded and installed by the user.

6.7.1 XPI File Structure

The structure of an XPI file is fairly simple. It contains an install manifest file (`install.rdf`), a chrome manifest file (`chrome.manifest`), default preferences (`defaults/preferences/` directory) and the chrome files (`chrome/` directory). The basic file structure can be seen in Listing 6.17.

```
extension.xpi
  install.rdf
  chrome.manifest
  /defaults
    /preferences
      preferences.js
  /chrome
    extension.jar
```

Listing 6.17: File structure of an XPI file.

6.7.2 Install Manifest

The install manifest file (`install.rdf`) stores all meta-data about the extension (see Mozilla [2006i]). It contains information about the author of the extension, the version, compatibility information and many more optional items. The basic layout can be seen in Listing 6.18. Commonly occurring elements include:

- *ID (`<em:id>`)*

  The unique ID of the extension. It is either a GUID or a string in the form `extensionname@some.domain`.

- *Version (`<em:version>`)*

  The version of the extension.

- *Target Application (`<em:targetApplication>`)*

  The target application for which the extension can be installed. The `<em:targetApplication>` tags contain an object giving more details about the application. The application itself is identified using a GUID (`<em:id>`). Additionally, minimum (`<em:minVersion>`) and maximum versions (`<em:maxVersion>`) of the application supported by the extension can be defined.

- *Name (`<em:name>`)*

  The name of the extension as it should be displayed in the user interface.

- *Description (`<em:description>`)*

  A short description of the extension, displayed in the extension manager user interface.

- *Creator (`<em:creator>`)*

  The name of the creator or lead developer of the extension.

- *Homepage URL (`<em:homepageURL>`)*

  A link to the extension's home page containing additional information about the extension.

- *Update URL (`<em:updateURL>`)*

  A link to the update manifest file (see Section 6.8). The file is periodically checked and new updates can be downloaded automatically.

- *Options URL (`<em:optionsURL>`)*

  A chrome URL to the extension's options dialogue box. If set, an options button is displayed in the extension manager interface.

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:em="http://www.mozilla.org/2004/em-rdf#">

  <Description about="urn:mozilla:install-manifest">
    <!-- Target Application - Thunderbird -->
    <em:targetApplication>
      <Description>
        <em:id>{3550f703-e582-4d05-9a08-453d09bdfdc6}</em:id>
        <em:minVersion>1.0</em:minVersion>
        <em:maxVersion>2.0.0.*</em:maxVersion>
      </Description>
    </em:targetApplication>

    <!-- Target Application - Seamonkey -->
    <em:targetApplication>
      <Description>
        <em:id>{92650c4d-4b8e-4d2a-b7eb-24ecf4f6b63a}</em:id>
        <em:minVersion>1.0</em:minVersion>
        <em:maxVersion>1.1.*</em:maxVersion>
      </Description>
    </em:targetApplication>

    <!-- Target Application - Mozilla -->
    <em:targetApplication>
      <Description>
        <em:id>{86c18b42-e466-45a9-ae7a-9b95ba6f5640}</em:id>
        <em:minVersion>1.7</em:minVersion>
        <em:maxVersion>1.8+</em:maxVersion>
      </Description>
    </em:targetApplication>

    <!-- Front End MetaData -->
    <em:id>{A23E4120-431F-4753-AE53-5D028C42CFDC}</em:id>
    <em:version>0.9.460</em:version>
    <em:name>ThreadVis</em:name>
    <em:description>Displays a small graphic visualising the context (thread) of the
        currently selected email.</em:description>
    <em:creator>Alexander C. Hubmann-Haidvogel</em:creator>
    <em:homepageURL>http://www.student.tugraz.at/ahubmann/threadvis/</em:homepageURL>
    <em:optionsURL>chrome://threadvis/content/SettingsDialog.xul</em:optionsURL>
    <em:updateURL>http://www.student.tugraz.at/ahubmann/threadvis/update.rdf</em:updateURL>
    <em:iconURL>chrome://threadvis/content/images/icon.png</em:iconURL>

    <!-- Packages, Skins and Locales that this extension registers -->
    <em:file>
      <Description about="urn:mozilla:extension:file:threadvis.jar">
        <em:package>content/</em:package>
        <em:locale>locale/en-US/</em:locale>
        <em:locale>locale/de-DE/</em:locale>
      </Description>
    </em:file>
  </Description>
</RDF>
```

Listing 6.18: A simple XPI install manifest file.

```
1 content  threadvis  jar:chrome/threadvis.jar!/content/
2
3 locale  threadvis  en-US   jar:chrome/threadvis.jar!/locale/en-US/
4 locale  threadvis  de-DE   jar:chrome/threadvis.jar!/locale/de-DE/
5
6 overlay  chrome://messenger/content/msgHdrViewOverlay.xul  chrome://
     threadvis/content/ThreadArcs.xul
7 overlay  chrome://communicator/content/pref/pref.xul  chrome://threadvis/
     content/Preferences.xul
8 overlay  chrome://messenger/content/preferences/preferences.xul  chrome
     ://threadvis/content/Preferences.xul
9 overlay  chrome://messenger/content/messenger.xul  chrome://threadvis/
     content/Status.xul
```

Listing 6.19: A simple XPI chrome manifest.

- *Icon URL (*`<em:iconURL>`*)*

  A chrome URL to an icon that should be displayed next to the name of the extension in the extension manager interface.

6.7.3 Chrome Manifest

The chrome manifest (see Section 6.3.3) stores information about all files that should be installed and registered in the chrome registry. It contains information about `content` packages, which are packages that should later be accessible by their own chrome URL. The `locale` packages store localised information. One `locale` package needs to be registered for each supported language. The `skin` package provides different user interface styles from which the user can choose from. The `overlay` keyword specifies overlays (see Section 6.1.5). By using the `style` keyword, style sheets can be applied to specific XUL files.

An example for a plain text manifest file is given in Listing 6.19. Line 1 uses a special syntax to register the `content` package. All chrome files are packed together in a JAR file (which is nothing else than a ZIP file). By using the `jar` keyword, Mozilla is instructed to unpack the JAR file on-the-fly when starting the application.

6.8 Updating Extensions

Mozilla provides a simple means of updating installed extensions to their newest versions (see Mozilla [2006a]). Each extension must provide a version string to identify its version as described in Mozilla [2006e]. This string consists of one or more version parts, separated by dots. Each part is itself a sequence of four optional parts, `<number-a><string-b><number-c><string-d>`, where `number` is any integer base 10, `string` is a simple ASCII string. So `1.2.3.4`, `1.5a` and `1.2pre3beta.2` are all valid version strings. When comparing versions, version parts are compared left to right, any empty or missing parts are equivalent to 0.

Using the manifest file `install.rdf` (see Section 6.7), an extension author can set the supported range of application versions using `minVersion` and `maxVersion` attributes (see Listing 6.20). Upon installing an extension, Mozilla compares its current version with the given range and refuses installation if the extension is not marked as compatible with the current version.

The `install.rdf` manifest file (see Section 6.7) can specify a URL to a special `update.rdf` file, which Mozilla periodically loads to check for new updates to the extension. The `update.rdf` file uses

a similar format to the `install.rdf` file, an example is given in Listing 6.20. The `em:updates` section contains a list of available updates as an RDF sequence. Each update specifies the version of the extension (`<em:version>`) and the compatible application (`<em:targetApplication>`). The application is identified using a GUID (`<em:id>`). The range of supported versions is defined using a minimum (`<em:minVersion>`) and a maximum version (`<em:maxVersion`). Finally, the link to the updated extension package is specified (`<em:updateLink>`).

6.9 Localisation

The goal of the Mozilla Localization Project [Mozilla, 2006d] is to make the translation of the user interface into a different language as simple as possible. Instead of hard-coding text directly into the application code, only a reference to a table of strings is used. This table can be altered by translators without the need to understand the application code. Mozilla [2006l] describes the process of localising an application based on Mozilla.

6.9.1 Localising XUL

As XUL is based on XML, XML entities can be used to provide localisation. The example below shows the use of a custom entity to define the label of a button. The label of the button will be replaced at run-time by the value of the entity `&entityname;`. For each supported language, a file containing the entity declarations needs to be created.

```
<button label="&entityname;"/>
```

Document Type Declaration (DTD) Files

Document Type Declaration (DTD) files are normally used to declare the syntax and semantics of XML files. In this case, they are only used to declare entities. Normally, one DTD file (with a .dtd extension) containing the localised text is created for each XUL file. To link an XUL file and a DTD file, a special declaration needs to be added to the XUL file. An example is given below:

```
<!DOCTYPE window SYSTEM "chrome://path/to/package/locale/xulfile.dtd">
```

Declaring Entities

The declaration of entities in a DTD file uses a simple syntax, as shown below. In an application, each label or string used in the interface would correspond to an entity defined in the DTD file. Additionally, entities can also be used to localise access keys or keyboard shortcuts.

```
<!ENTITY entityname "entityname is replaced with this text wherever it is
    used.">
```

6.9.2 Localising Scripts

In scripts, entities can not be used, since script files are not parsed for entities. In this case, property files (with a .properties extension) have to be used. Property files contain sets of strings in a name = value syntax, one definition per line. An example of a property file is given in Listing 6.21.

To read the properties, XUL provides the `stringbundle` element. This element reads the property file and builds a list of the strings, which can be accessed by scripts. The example below shows the declaration of such an element:

```
<?xml version="1.0"?>
<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:em="http://www.mozilla.org/2004/em-rdf#">

  <RDF:Description about="urn:mozilla:extension:{A23E4120-431F-4753-AE53-5D028C42CFDC}">
    <em:updates>
      <RDF:Seq>
        <RDF:li>
          <RDF:Description>
            <em:version>0.9.460</em:version>
            <!-- Target Application - Thunderbird -->
            <em:targetApplication>
              <Description>
                <em:id>{3550f703-e582-4d05-9a08-453d09bdfdc6}</em:id>
                <em:minVersion>1.0</em:minVersion>
                <em:maxVersion>2.0.0.*</em:maxVersion>
                <em:updateLink>http://www.student.tugraz.at/ahubmann/threadvis/ThreadVis.xpi
                    </em:updateLink>
              </Description>
            </em:targetApplication>
            <!-- Target Application - Seamonkey -->
            <em:targetApplication>
              <Description>
                <em:id>{92650c4d-4b8e-4d2a-b7eb-24ecf4f6b63a}</em:id>
                <em:minVersion>1.0</em:minVersion>
                <em:maxVersion>1.1.*</em:maxVersion>
                <em:updateLink>http://www.student.tugraz.at/ahubmann/threadvis/ThreadVis.xpi
                    </em:updateLink>
              </Description>
            </em:targetApplication>
            <!-- Target Application - Mozilla -->
            <em:targetApplication>
              <Description>
                <em:id>{86c18b42-e466-45a9-ae7a-9b95ba6f5640}</em:id>
                <em:minVersion>1.7</em:minVersion>
                <em:maxVersion>1.8+</em:maxVersion>
                <em:updateLink>http://www.student.tugraz.at/ahubmann/threadvis/ThreadVis.xpi
                    </em:updateLink>
              </Description>
            </em:targetApplication>
          </RDF:Description>
        </RDF:li>
      </RDF:Seq>
    </em:updates>
    <em:version>0.9.460</em:version>
    <em:updateLink>http://www.student.tugraz.at/ahubmann/threadvis/ThreadVis.xpi</
        em:updateLink>
  </RDF:Description>
</RDF:RDF>
```

Listing 6.20: Example `update.rdf` manifest file.

```
name=value
alert=This is an alert.
error=An error occurred.
```

Listing 6.21: Declaration of properties in a property file.

```
<stringbundle id="strings" src="strings.properties"/>
```

The element `strings` can be used to access the properties stored in the file `strings.properties`. The code below shows how scripts can access the `stringbundle` element and read the properties stored inside a properties file:

```
var strbundle = document.getElementById("strings");
var error = strbundle.getString("error");
```

6.10 Conclusion

Developing extensions for Mozilla-based applications is very similar to developing web applications: XUL very much resembles HTML, the styling of user interface elements even uses CSS and application code is implemented in JavaScript. Using overlays makes adding new elements easy, as there is no need to change existing code. It is even possible for multiple extensions to change and extend the same element.

Package management and integrated, automatic updates help both users and developers in distributing and updating extensions. Providing an extension in many languages is supported by advanced localisation and internationalisation features. Since nearly all functionality is provided by means of simple JavaScript objects, an extension developer does not need to care about pointers or memory management. This makes writing advanced, interactive extensions as easy as writing a dynamic web application.

This development framework has inspired thousands of developers, the official Mozilla add-on portal [Mozilla, 2008b] lists over 300 add-ons for Mozilla Thunderbird and 2000 add-ons for Mozilla Firefox. Add-ons exist for nearly all necessary and unnecessary features missing from a standard Thunderbird installation.

Chapter 7

ThreadVis for Mozilla Thunderbird

> *"All but one of the users said that they would like a thread visualization in their future email clients."*
>
> [Kerr [2003a]]

7.1 Overview

ThreadVis is a Mozilla Thunderbird extension, implementing the Thread Arcs idea described in Kerr [2003a] (see Section 5). Being an extension (for technical details about Mozilla extension development see Section 6), the ThreadVis extension can be easily installed in any existing Thunderbird installation with no additional software necessary. Installation is as easy as downloading the extension XPI file and opening the file in Thunderbird. After installation, the new visualisation user interface element is added to the standard layout using previously unused space inside the mail client. Figure 7.1 shows the Mozilla Thunderbird mail client with the ThreadVis extension installed.

The extension runs in the background, analysing necessary email header information and providing the user with a Thread Arcs like visualisation for the selected message. An extensive cache speeds up the calculation and the display of the visualisation. This is particularly useful as all program logic is implemented in JavaScript, which helps making the extension highly portable, but also leads to less than ideal performance for computing intensive tasks.

Using the integrated automatic update mechanism, users of the extension are informed whenever a new version of the extension is available. Thunderbird then downloads the new version and installs it without any user interaction necessary.

7.2 Using the Extension

After the ThreadVis extension is installed and Thunderbird is restarted, the user is greeted with a simple wizard describing the functionality of the extension and giving the user the possibility to configure its behaviour (see Section 7.3 for details). On the first start of the extension, a cache containing information about all threads in the user's mailbox needs to be created to speed up later calculations. This step can take an extremely long time for larger inboxes. After this initial computation, only newly arrived messages need to be analysed and stored in the cache. This helps in keeping the effect of the extension on the normal behaviour of the mail client to an absolute minimum.

When the user selects a mail message in Thunderbird, ThreadVis fetches thread information from the previously generated cache. All messages from the corresponding thread are analysed and the Thread

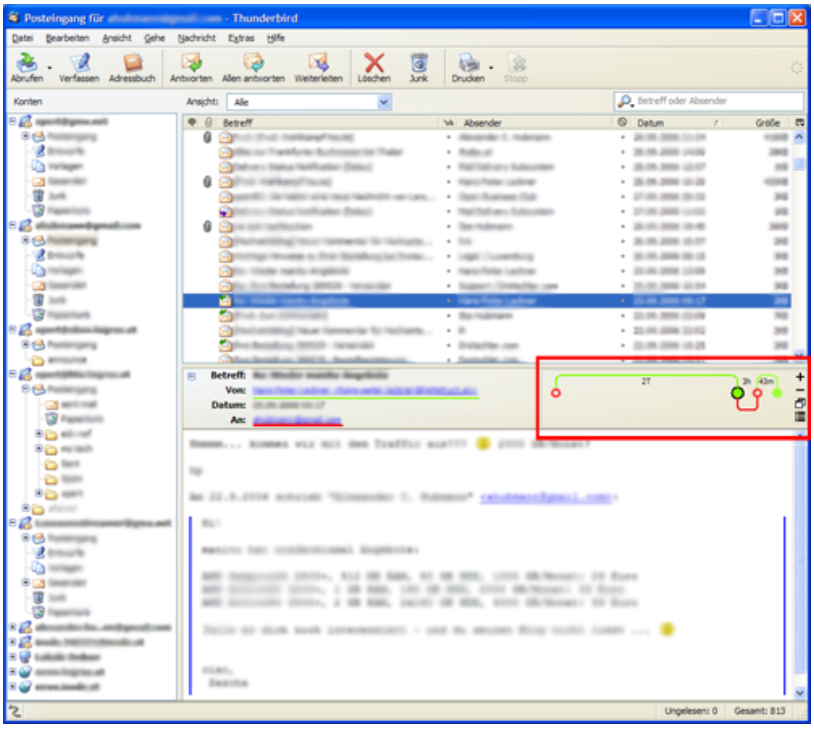


Figure 7.1: The ThreadVis extension in Mozilla Thunderbird. Message details have been blurred for privacy reasons.

Arcs visualisation is displayed. The selected message is highlighted to give the user information about the chronological position of the message relative to other messages in the conversation. The visualisation automatically adjusts its size to the available space and zooms out for large threads to accommodate the whole visualisation. The user can zoom in and pan the visualisation if it becomes larger than the available viewport. If a message is selected from outside the viewport, the visualisation automatically pans to make the selected message visible.

ThreadVis can also be used as an input element, providing the user with an additional means to navigate to specific messages in the thread, by clicking on the corresponding visualisation element for that message. This makes navigating the messages of a conversation far easier than scanning through the inbox and manually searching for the appropriate mails. The extension even supports seamless switching between different folders (e.g. the inbox and the sent-mail folder) while displaying the messages of a conversation. In addition, a small tool tip is displayed when hovering above the visualisation element, showing basic information such as the name of the author, the message date and the subject line.

For extremely large conversations, a pop-up window containing the visualisation can be opened, which displays the visualisation outside the normal Thunderbird user interface. To obtain more details about the persons involved in a conversation, a separate legend window can be opened which lists all contributors to a thread, their assigned colour, and the number of messages the person contributed to the conversation.

7.3 User Settings

The ThreadVis extension provides a large number of user-definable settings that affect the behaviour and the layout of the visualisation. Mozilla Thunderbird provides a preferences framework which allows an extension to supply a list of configurable options together with the default values. ThreadVis extends the standard options dialogue and adds its own section, containing seven tabs.

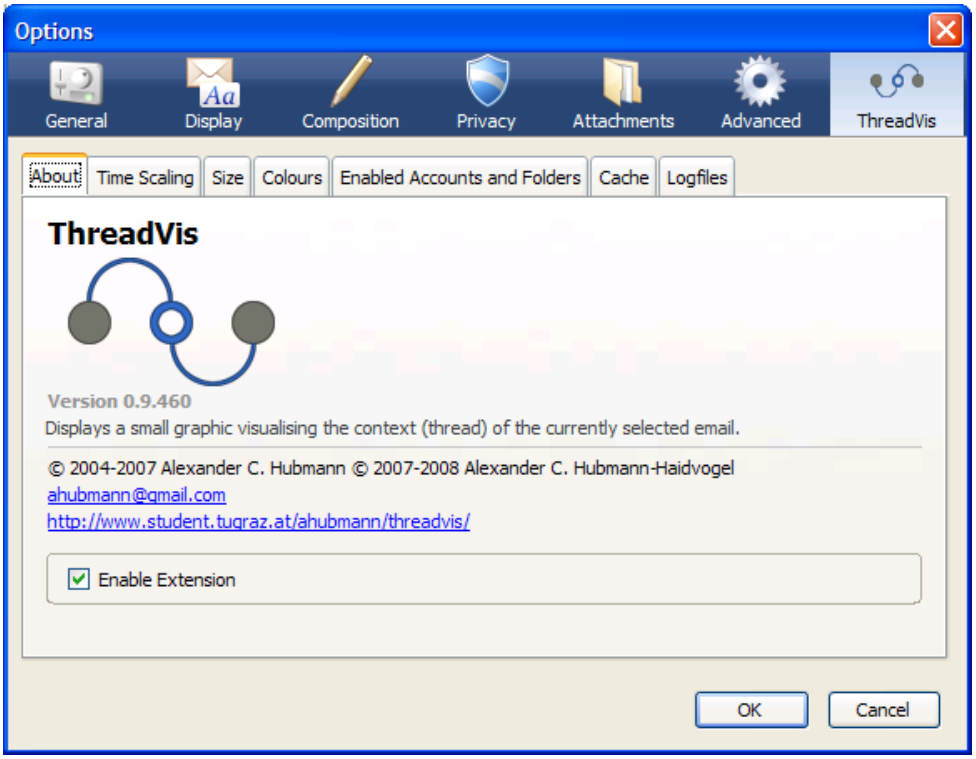


Figure 7.2: The ThreadVis About tab.

7.3.1 About

The first tab (see Figure 7.2) consists of a short About section, displaying the version number of the ThreadVis extension, a link to the web site and a contact email address. Additionally, the user can enable or disable the extension. Mozilla Thunderbird already supports enabling and disabling certain extensions, but the mail client needs to be restarted for changes to become effective.

7.3.2 Time Scaling

Time Scaling (see Section 7.8.5) can be enabled or disabled. An additional Time Line displaying more information about the time between messages can also be enabled in this dialogue (see Figure 7.3).

7.3.3 Size

The Size tab (see Figure 7.4) gives the user many possibilities to tweak the visualisation. The size of the messages directly affects the diameter of the circles representing emails. The minimal arc size specifies the height of the first (lowest) arc. Setting the arc radius affects the corners of the arcs, setting it to 0 leads to rectangular arcs. The vertical spacing between two arcs can also be set, as well as the line width used to draw an arc. The minimal spacing between two messages also affects the overall size of the visualisation if Time Scaling is enabled, since the horizontal size is then calculated proportional to this minimal size (see Section 8.3).

The final setting, the maximum size of the visualisation, allows the user to indirectly set the default zoom level. By default, the visualisation is rendered to fit the size of the visualisation box. Setting a larger size here increases the size of the visualisation and panning is needed to display the whole visualisation.

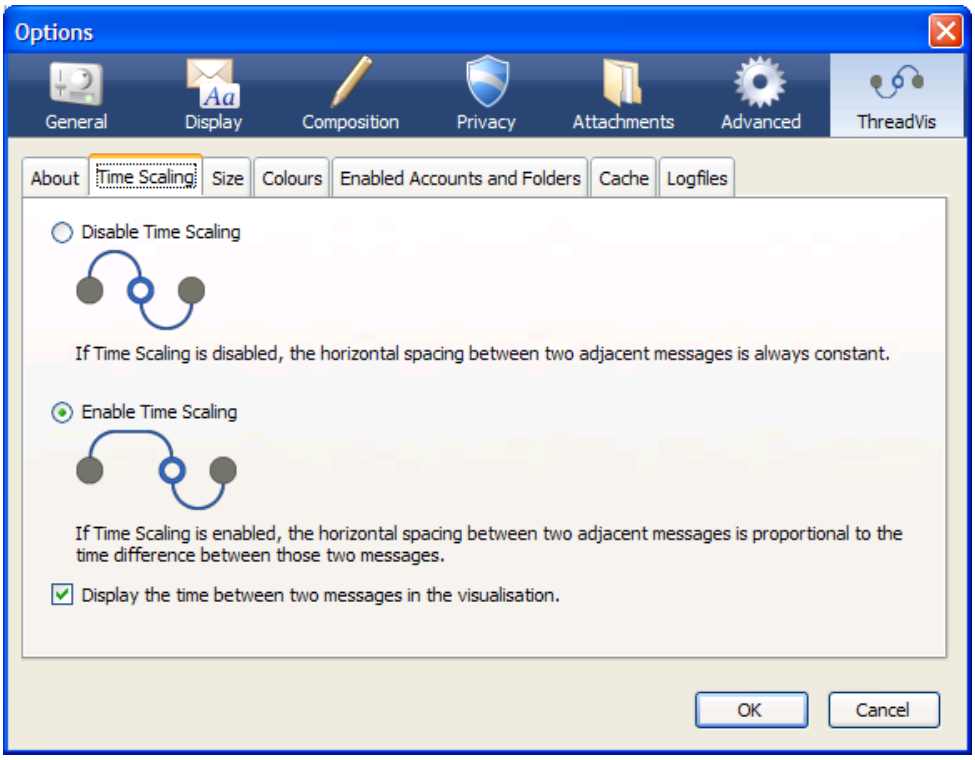


Figure 7.3: The ThreadVis Time Scaling tab.

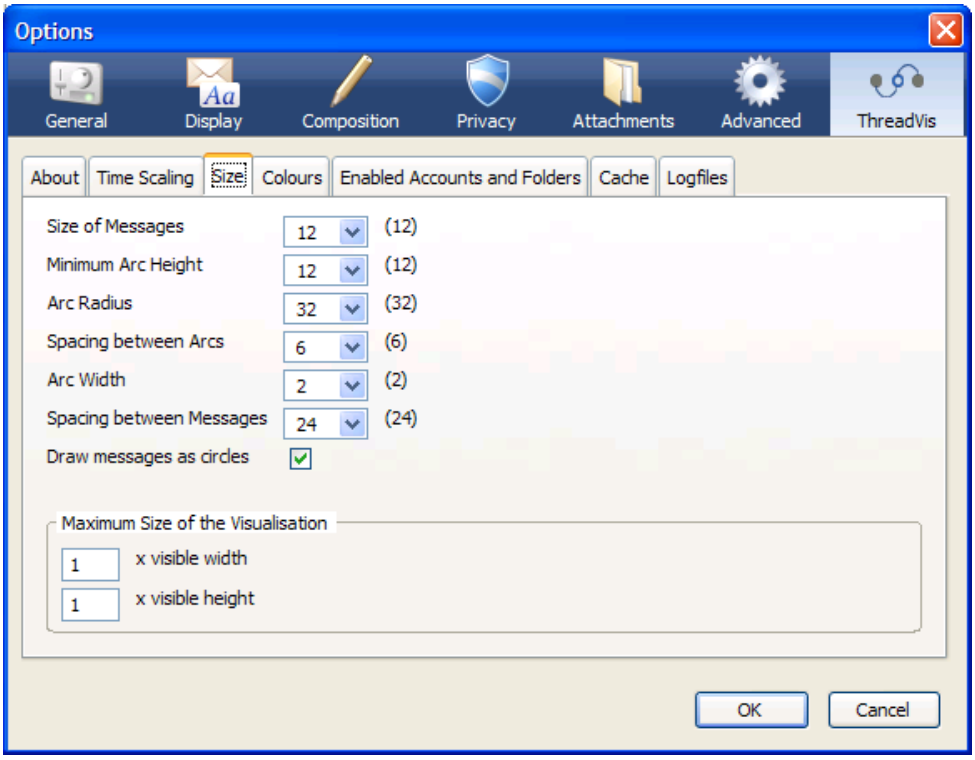


Figure 7.4: The ThreadVis Size tab.

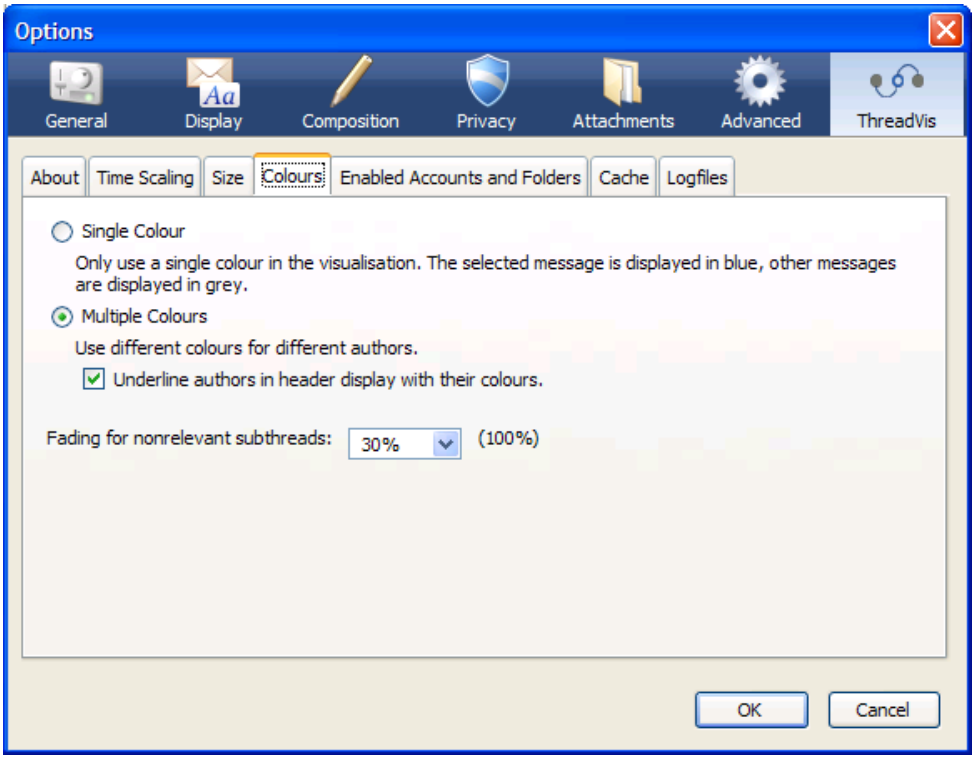


Figure 7.5: The ThreadVis Colours tab.

7.3.4 Colours

The user can select whether the extension should only use two colours to visualise a conversation or if multiple colours should be used (see Figure 7.5). If only two colours are used, a blue message denotes a message sent by the user, a grey message represents a received message.

If multiple colours are used, a new colour is created for each contributor to the thread. Additionally, the names and email addresses of authors can be underlined in the header view to associate the colours with the contributors. For a detailed overview of all authors of a thread, a legend can be displayed (see Section 7.8.2). The user is also given the option to fade out unrelated subthreads, which decreases the saturation of the colours used to display unrelated parts of the conversation.

7.3.5 Enabled Accounts and Folders

To give the user more control over the analysed messages, the accounts and folders for which ThreadVis should be enabled can be selected (see Figure 7.6). The extension can either be totally disabled for a specific account or only for certain folders. Disabling ThreadVis for certain folders can speed up the calculation of conversations and avoid unnecessary calculations. For example, ThreadVis can be disabled for junk mail folders or for folders storing newsletters as such messages do not typically form conversations.

7.3.6 Cache

The Cache tab (see Figure 7.7) simply allows the user to clear the thread cache for a particular account. This is useful if the calculation produced incorrect results or if the cache became corrupted and needs to be rebuilt.

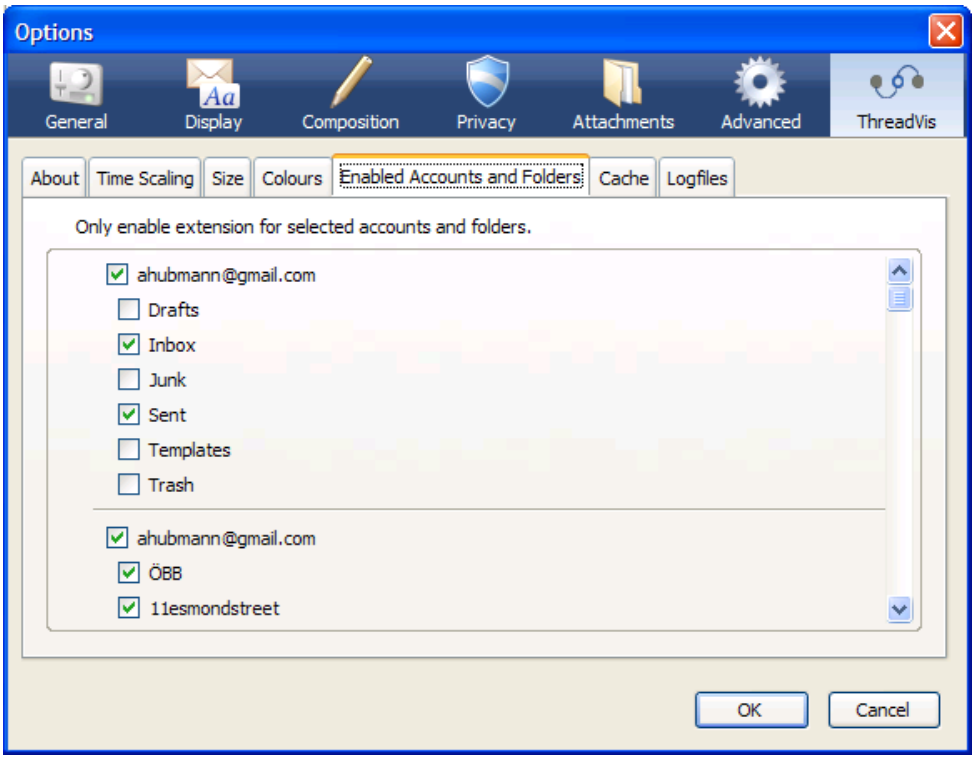


Figure 7.6: The ThreadVis Enabled Accounts and Folders tab.

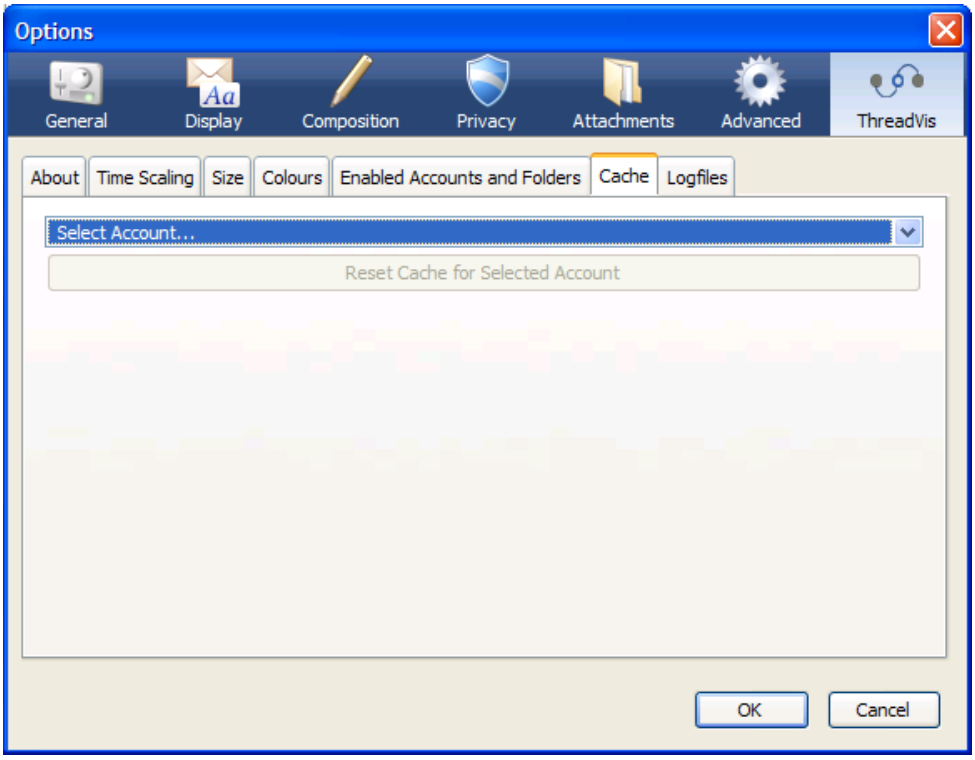


Figure 7.7: The ThreadVis Cache tab.

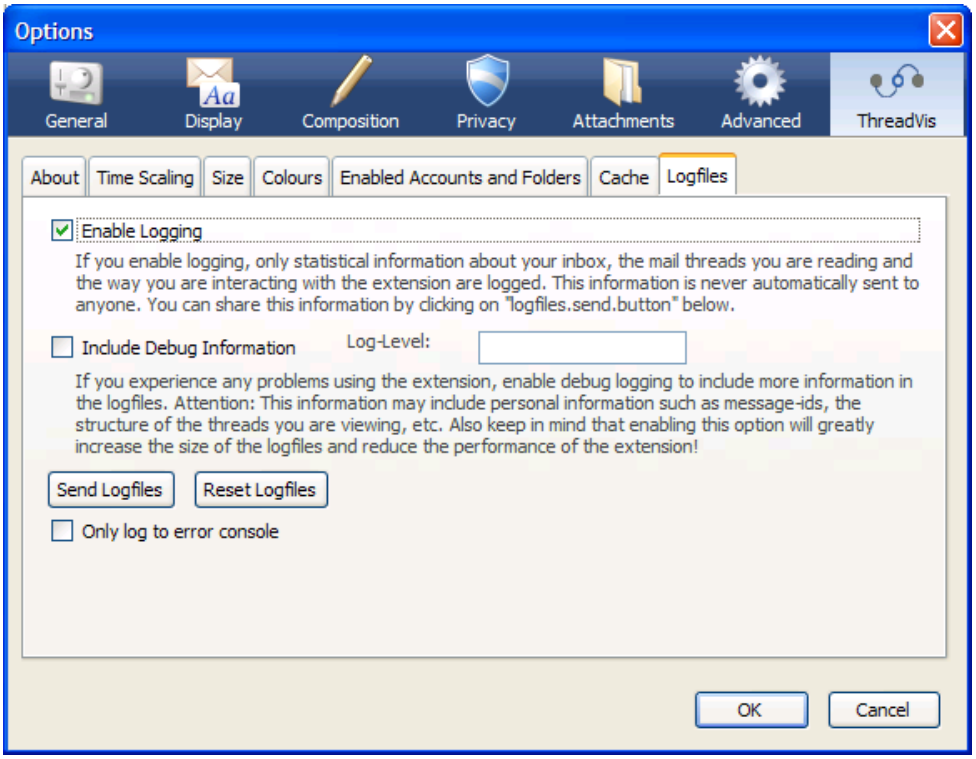


Figure 7.8: The ThreadVis Logfiles tab.

7.3.7 Logfiles

This tab provides basic settings to enable or disable the built-in logging feature (see Figure 7.8). A usage study was conducted as part of ThreadVis development. For the usage study, anonymised usage log files were returned by participants to the developer (see Section 9). To simplify debugging, extensive debug information for particular modules can be included in the log file. To allow simple feedback, a single button click automatically composes a mail message to the developer and attaches the created log file. A second button allows the user to delete the created log file.

7.4 Implementation

As already discussed in Section 6, Thunderbird uses XUL files to define the basic user interface elements and JavaScript to provide interactivity. Access to Thunderbird functions and email data is provided either by global JavaScript objects or by Scriptable Components (see Section 6.5), which wrap underlying C++ objects to JavaScript. The ThreadVis extension was implemented entirely in JavaScript and supports any platform which Thunderbird supports. Using the chrome system, the user interface elements for the visualisation are added to the standard mail user interface. By using overlays (see Section 6.1.5), additional user interface elements can be added without needing to edit the original source code, which greatly simplifies installing and uninstalling extensions.

7.5 Goals

One of the main goals in implementing the ThreadVis extension was to provide users with an additional means to navigate their mailbox while being as minimally invasive as possible in terms of user interface, performance, and usability. ThreadVis should extend and enhance the normal user interface, but avoid

adding too much clutter or distracting the user from normal operations. Activating ThreadVis should have little or no impact on performance in normal day-to-day operations. Finally, the usability of the mail client should be improved, but existing functionality and behaviour should not be changed or blocked.

By using previously unused space inside the user interface, the extension does not interfere with any existing elements. Using extensive caching techniques, performance issues have been reduced. Only at the very first startup, where all existing messages need to be scanned, are normal operations in the client slowed down. By doing the analysis in the background, the extension tries to keep this impact on performance as low as possible.

7.6 Obstacles

The main obstacle in implementing ThreadVis for Thunderbird was the programming language and the provided API to core Thunderbird functions. As JavaScript is an interpreted language, great care is needed in writing fast code. The performance of the JavaScript interpreter used in Mozilla is fairly good, the main downside of the implementation is the single-threaded user interface.

Only a single thread is used to draw the user interface elements and to execute JavaScript code used both in Mozilla directly and by extensions. This means that a time-consuming and processor-intensive JavaScript operation can block the whole user interface. Thus, great care is needed to avoid such blocks and to not disrupt the user experience by slowing down the mail client.

Avoiding blocking methods can be tricky. JavaScript allows for semi-threaded operation by using `setTimeout` calls that asynchronously call methods with a certain delay. To avoid blocking the user interface, long running methods need to be split up into shorter elements and execution time has to be yielded to other user interface elements. As JavaScript does not implement any `yield` methods or any other means to influence the scheduling, workarounds have to be implemented. Using a `setTimeout` call with a delay of zero milliseconds does not delay the execution, but has the side-effect of yielding the execution to other JavaScript methods (and thus also the main Mozilla Thunderbird user interface code).

7.7 Caching

Early versions of ThreadVis used on-the-fly calculation of message conversations. This scaled well to a maximum of around 5000 messages in a single account, since all messages needed to be scanned on each start-up of the program. Threading more than 5000 messages took considerably more time and slowed down the overall performance of the mail client. Introducing an extensive caching architecture, the calculation of threads could be sped up considerably.

Using an event-driven model, newly arrived messages trigger a cache update. The headers of the arrived messages are analysed and their parent messages are looked up in the user's mailbox. After the complete conversation for the message has been found, information about all messages in the thread is stored in the local copy of the message's header.

When visualising a message, ThreadVis does not need to scan the whole inbox, but needs only to look at the messages stored in the cache and loads only messages actually contained in the conversation. That way, ThreadVis is able to scale to practically any mailbox size as only the relatively small number of newly arrived messages have to be analysed and threaded.

7.8 The ThreadVis Visualisation

ThreadVis displays a Thread Arcs [Kerr, 2003a] like visualisation in the Thunderbird user interface. Using previously unused screen space, the visualisation does not interfere with normal user interaction.

(a) (b)

Figure 7.9: Visualising the same conversation. (a) Visualisation using only two colours, one for sent and one for received messages. (b) Using a unique colour for each contributor.

Colour coding helps distinguishing different contributors to the discussion, a time line and Time Scaling emphasise chronology. Fading out unrelated messages helps the user to focus on a particular sub-thread of the discussion.

7.8.1 Colours

The ThreadVis visualisation can be drawn using only two colours (as described in [Kerr, 2003a]) using the first colour to display messages sent by the user and a second colour for all received messages. To further differentiate the two message groups, sent messages are visualised hollow, received messages full.

Using multiple colours as in Figure 7.9, every contributor to the conversation can be assigned a unique colour. That way, conversations with relatively few contributors can easily be distinguished from threads with many contributors. A legend (see next section) displays the mapping between colours and authors and ThreadVis can also underline all known authors in the header view with the corresponding colour from the visualisation.

7.8.2 Legend

As larger threads often contain mails from many participants, the colour coding (see previous section) of messages can become unclear. In this case, the user can open a legend window (see Figure 7.10), which shows the mapping of thread participants to assigned colours.

The legend lists the colour assigned to each person, their name and email address and, additionally, the number of messages contributed to the thread. As already discussed in the previous section, the names of all thread contributors listed in the header view are automatically colour-coded. For larger threads, the header view (the `From`, `To`, `CC` and `BCC` headers) rarely contains all persons involved in the thread. In those cases, the legend window provides a list of all contributors to the user.

7.8.3 Focus on Sub-Thread

Inside a single conversation, several sub-threads covering largely unrelated topics can emerge. To focus on the appropriate sub-thread for the selected message, the visualisation can fade-out unrelated sub-threads that do not belong to the current message. When displaying an email, all parent messages and all child messages are displayed fully, all other messages are faded by a selectable amount. Figure 7.11 shows a long conversation split into several sub-threads. Figure 7.11a shows the first (parent) message selected in the mail client and all its children and thus the whole conversation is displayed fully. In Figure 7.11b, a message nested in a sub-thread is selected. By selectively fading-out the non-related messages, the user is able to obtain a better overview of related messages. Additionally, the link between the selected message and the root message (the first message of the thread) is immediately visible as all messages between those two messages are in focus and clearly visible and all other unrelated messages are faded out.

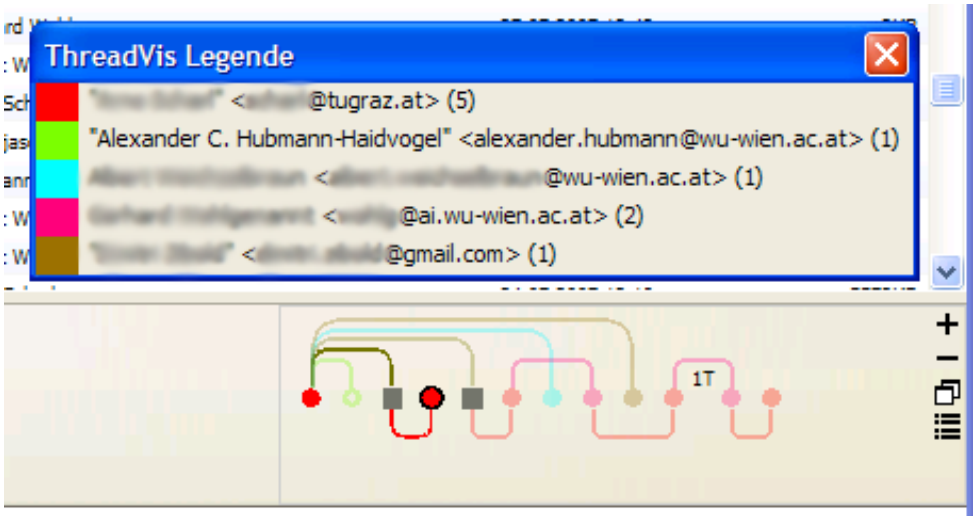


Figure 7.10: Displaying a legend window for a thread visualisation showing the colours for different contributors. Blurring has been used to remove private details.

(a) **(b)**

Figure 7.11: Fading out unrelated parts of a large conversation helps give a better overview over related messages. In (a), the first message is selected, in (b) the last message is selected (enclosing black circle). Note that the link between the selected message and the root message is immediately obvious.

(a) (b)

Figure 7.12: Visualising the same conversation. (a) Normal visualisation. (b) Visualisation using Timeline.

(a) (b)

Figure 7.13: Visualising the same conversation. (a) Normal visualisation. (b) Visualisation using Time Scaling. It is immediately obvious that the second reply took far less time than the first reply.

7.8.4 Timeline

The Thread Arcs visualisation already provides insight into the chronological order of all messages of the conversation, but it gives no hint about the time difference between two messages. Without any additional information, a visualised conversation might have taken three hours or five weeks. By activating the Timeline feature, the user can display the time difference between two messages using a short text note. Figure 7.12 shows the same conversation, using the normal Thread Arcs visualisation in (a) and providing additional information using the Timeline in (b).

7.8.5 Time Scaling

Time Scaling expands the Timeline idea by giving the user additional clues about the time passed between two messages. Normally, messages in the Thread Arcs visualisation (as presented in [Kerr, 2003a]) are displayed equally spaced horizontally. By enabling Time Scaling, the horizontal spacing between two messages is proportional to the time passed between those messages. Section 8.3 gives more details on the implementation of this feature.

Figure 7.13 shows two visualisation for the same four messages: (a) shows the normal Thread Arcs visualisation, giving the user no details about the time passed between the messages, (b) shows the same messages with Time Scaling enabled. It is immediately obvious that the second reply took far less time than the first reply. Time Scaling gives less detail than the Timeline, but is more obvious and intuitive.

7.9 Conclusion

The ThreadVis extension provides a Thread Arcs (see Section 5) like visualisation for any Mozilla Thunderbird mail client. It improves the integrated threading view of the client, which cannot handle threads with messages stored in multiple folders, such as the inbox and the sent-mail folder. The visualisation extends the user interface of the mail client and adds a small visualisation in the expanded header view pane, using previously unused screen space. ThreadVis provides an additional means to navigate between messages in a thread, without affecting the normal behaviour of the program.

An extensive caching architecture and optimized algorithms keep the necessary calculations to a minimum, to have as little impact on performance as possible. Additionally, folders such as the junk mail or drafts folder can be excluded from the calculations. In contrast to the built-in thread view, ThreadVis

allows users to keep their inbox sorted by date and adds thread information in a separate visualisation. For large threads, the visualisation can be displayed in a pop-up window, a legend provides an overview of the participants of the thread.

Chapter 8

Selected Details of the Implementation

> *"In general, an implementation should be conservative in its sending behavior, and liberal in its receiving behavior."*
>
> [Postel [1980]]

This chapter gives an overview over selected details of the implementation of the ThreadVis extension for Mozilla Thunderbird. The examples show the use of the JavaScript programming language and the usage of XUL (see Section 6.1) to place the user interface elements.

First, the algorithm used to find conversations in the user's mailbox is shown. Although Thunderbird already implements a threading algorithm, it had to be re-implemented in the extension due to limitations in the native code. The next section discusses the use of XUL to visualise the conversations. Using cascading style sheets (CSS) user interface elements can be styled to form the thread arcs used to display the email conversation. The third section describes the Time Scaling algorithm (see Section 7.8.5). The normal Thread Arcs visualisation only displays the chronology of the messages in the conversation. Time Scaling additionally visualises the elapsed time between messages and enhances the temporal information displayed. The caching algorithm (see Section 7.7) is discussed in the last section. To avoid re-analysing all messages over and over again, this algorithm stores information about the messages forming a thread.

8.1 Threading

Mozilla Thunderbird already provides a basic means to display messages in a threaded view. However, the threading code only analyses messages of a single folder and thus is primarily useful for grouping newsgroup messages. As email messages are typically stored in several folders (in typical setups, at least an inbox and a sent-mail folder), this approach is not well-suited to thread emails. One possible solution is to store all mails in a single folder (e.g. mail all outgoing emails to oneself so that sent mails end up in the inbox). As can be easily seen, this approach is no solution since it does not allow the user to use folders in a standard way.

To overcome this limitation, the threading code was re-implemented in JavaScript. The ThreadVis extension analyses all messages of a single account regardless of the folder they are stored in. This allows the calculation of threads across folders, giving the user the freedom to use any number of folders to store messages.

Since the calculation of a thread is time-consuming (see below), extensive caches are used so that once a message has been associated with a particular thread, the message does not have to be analysed again. To group messages together in a thread, email header information such as the In-Reply-To header and the References header (see RFC 2822 [Resnick, 2001]) need to be analysed:

> The “In-Reply-To:” and “References:” fields are used when creating a reply to a message. They hold the message identifier of the original message and the message identifiers of other messages (for example, in the case of a reply to a message which was itself a reply). The “In-Reply-To:” field may be used to identify the message (or messages) to which the new message is a reply, while the “References:” field may be used to identify a “thread” of conversation.

These headers can obviously only point to mails in the past, they represent a one-way relationship between messages. Thus, the most recent email of a thread points to the other messages in the conversation, whereas the first message which started the thread holds no information about the conversation whatsoever.

To overcome this limitation, the reference fields of all mails need to be traversed backwards to collect all messages that form a conversation. Once this information is calculated for all messages in a particular account, a new private header field (X-ThreadVis-Cache) is added to all analysed messages containing a list of all messages in the conversation. That way, all messages in a conversation can be retrieved, regardless of which email is currently displayed in the mail client.

When analysing a new message that has no cache information attached, emails that are linked in the reference fields are fetched from the users account until a message with attached cache information is found. The new message is then added to the conversation and the cache for all messages is updated. This allows the extension to obtain all messages of a particular thread in constant time, regardless of the size of the user’s inbox.

Listing 8.1 shows the threading code used by the extension. At first, all messages are analysed and indexed by message id. For each entry in the reference fields described above, the corresponding message is retrieved. As the reference field lists the emails in chronological order, the messages of the thread can be linked in this order. By iteratively linking the referenced messages, the whole conversation of a message can be reconstructed.

If the user’s mailbox contains two messages with the same message id, this most probably means that a message the user sent also arrived in the inbox. In this special case only the sent mail is stored, since it should be visualised as a sent email later on. After all messages are analysed, the root set is calculated. The root set stores all messages that do not have a parent message and thus represent the first email of a thread.

8.2 Visualisation using XUL and Cascading Style Sheets

As Mozilla Thunderbird uses the XUL user interface language (see Section 6.1), visualisation elements can be drawn using Document Object Model (DOM) [W3C, 2008b] elements. Similar to HTML documents, those elements can be styled using Cascading Stylesheets (CSS) [W3C, 2006]. Using an object-oriented programming model, the elements are displayed and drawn in the user interface from the previously calculated conversation elements. When navigating messages in the same thread, existing visualisation elements are re-used to speed up the drawing. This approach scales well to approximately 50 messages, above which the initial drawing becomes too slow to be usable.

Scalable Vector Graphics (SVG) [W3C, 2008c] would be an ideal candidate for the visualisation. Unfortunately, current Thunderbird versions cannot display SVG documents as user interface elements. As soon as such builds become available, the visualisation code could be easily adapted to support SVG.

The current implementation uses simple box elements to visualise the conversation. Each message is represented by a single box which is styled using CSS properties. The Mozilla-specific `-moz-border-radius` (similar to the CSS 3 property `border-radius` [W3C, 2008a]) is used to style the normally square box as a circle representing one message. By selectively setting the background property of the box, full and hollow circles can be drawn to represent incoming and outgoing messages. Similarly, the

```
1  // store all containers by messageId
2  idTable[]
3
4  // put all messages in containers
5  foreach message {
6      // try to get already stored container
7      messageContainer = idTable[messageId]
8
9      // if a container already exists
10     if (messageContainer) {
11         // check if stored message container is empty or not a sent-mail
12         // it may be empty because we created an empty container
13         // while looking at the references header
14         // two messages with the same messageId can exist,
15         // if a user sends a message to himself, causing the mail to
16         // end up in the inbox and the sent-mail folder
17         // as it is a sent-mail, we want to keep this reference
18         if (messageContainer.isEmpty() || ! messageContainer.isSent()) {
19             // set message in container
20             messageContainer.setMessage(message)
21             idTable[messageId] = messageContainer
22         }
23     }
24
25     // if no suitable container found, create new one
26     if (! messageContainer) {
27         messageContainer = new Container()
28         messageContainer.setMessage(message)
29         // index container by messageId
30         idTable[messageId] = messageContainer
31     }
32
33     // for each element in references field of message
34     for (reference in references) {
35
36         // try to find container for referenced message
37         referenceContainer = idTable[referenceId]
38         if (referenceContainer == null) {
39             // no container found, create new one
40             referenceContainer = new Container()
41             // index container
42             idTable[referenceId] = referenceContainer
43         }
44
45         link reference containers together
46     }
47 }
```

Listing 8.1: Pseudo-code for the ThreadVis threading algorithm.

arcs connecting the messages are boxes with unused borders set to zero. The currently selected message is highlighted using a black circle drawn around the message box. To make selecting a message easier, a separate larger, clickable, but otherwise invisible element is drawn around each message.

Browsing messages in the same conversation does not normally trigger a full refresh of the visualisation. In the simplest case, only the marker for the currently selected message has to be redrawn. If the size of the viewport changes, the whole visualisation has to be rescaled. Rather than deleting all elements, the position and size of the elements is adapted to the new size. This further speeds up redrawing of the same thread.

Listing 8.2 shows an example of a simple visualisation of five messages. Each mail visualisation consists of three `box` elements, the first visualises the message itself, the second draws the black circle to highlight the message, and the third is a larger box to catch clicks intended for the message. Thread arcs between the messages are also drawn using `box` elements, with the top or bottom border hidden depending on the position of the arc. Each message also has an associated tooltip element to show information about the message, which is filled in dynamically. The last four `description` elements display the timeline (see Section 7.8.4) for the visualisation.

8.3 Time Scaling

The visualisation described in Kerr [2003a] uses fixed horizontal spaces between messages, regardless of the time passed between two messages. Time Scaling enhances the visualisation by giving the user additional clues about the time difference between messages. It spaces the messages proportional to their time difference using as much space as possible.

When calculating the visualisation parameters, all messages of the conversation are analysed and the time between the mails is stored and the minimum difference calculated. This minimum time difference is then mapped to the minimal horizontal distance as defined in the user settings. For all other time differences, the horizontal distance is calculated proportional to this minimal distance. This first pass lays out the conversation but does not consider the available screen space.

A second pass then fits the visualisation to the available viewport. It does so by repeatedly reducing the previously calculated spacings by a common factor (in the current implementation 0.9) until the visualisation fits the available width or all spacings have been reduced to the minimal spacing mentioned above. This iterative algorithm ensures that the spacing between to messages is proportional to the time difference and that, at the same time, the difference never falls below the minimal spacing. This is the optimal behaviour as all the available horizontal space is used to accommodate the message visualisation. If enough space is available, the messages with the greatest time difference are spaced farthest apart. If the horizontal space is not sufficient, the differences are iteratively reduced until the visualisation fits the available space or all differences are at the minimum difference as defined above. This means that the Time Scaling implementation gracefully degrades to the normal visualisation as described in [Kerr, 2003a] when not enough space is available.

Listing 8.3 shows the described algorithm as pseudo-code. The first pass (lines 1-3) simply calculates the time difference `t` between all messages and computes the minimal difference `tMin`. In lines 5-9, the needed horizontal spacing `s` for every message is calculated proportionally to the minimal difference `t` from step 1. Additionally, the total width `sNeeded` of the visualisation is calculated. `sAvailable` stores the maximum available spacing from the available width for the visualisation `maxWidth` and the minimal spacing between two messages `sMin`. The loop in lines 15-21 then reduces the spacing for each message by a factor of 0.9 if the needed spacing `sNeeded` exceeds the available spacing `sAvailable`. The abort condition on line 20 ensures that the loop exits if the spacing between all messages has been reduced to the minimal spacing.

Finally, Listing 8.4 shows the Time Scaling code as it is implemented in the ThreadVis extension. The input parameters for the method are the list of containers (`containers`), the previously calculated

```
1
2  <stack id="ThreadVisStack">
3      <box style="top: 23.93px; left: 4.79px; width: 9.58px; height: 9.58px; -moz-border-
           radius-topleft: 9.58px; ... cursor: default;"/>
4      <box style="border: 1.59px solid black; top: 22.33px; ... -moz-border-radius-topleft:12
           .76px; ..." hidden="true"/>
5      <box style="top: 19.14px; ... vertical-align: top;" tooltip="ThreadVis_12" context="
           dot_popup_12"/>
6      <tooltip orient="vertical" id="ThreadVis_12"/>
7      <popupset>
8          <popup id="dot_popup_12"/>
9      </popupset>
10     <box style="top: ... -moz-border-radius-topleft: 9.57143px; ..."/>
11     <box style="border: 1.59px solid black; top: ... -moz-border-radius-topleft: 12.76px;
           ..." hidden="true"/>
12     <box style="top: ..." tooltip="ThreadVis_210" context="dot_popup_210"/>
13
14     <tooltip orient="vertical" id="ThreadVis_210"/>
15     <popupset>
16         <popup id="dot_popup_210"/>
17     </popupset>
18     <box style="border-top: 1.59px solid rgb(127, 255, 0); border-left: ... top: ... -moz-
           border-radius-topleft: 32px; -moz-border-radius-topright: 32px;"/>
19     <box style="top: ... opacity: 0.3; -moz-border-radius-topleft: 9.57px; ..."/>
20     <box style="border: 1.59px solid black; top: ... -moz-border-radius-topleft: 12.76px;
           ..." hidden="true"/>
21     <box style="top: ..." tooltip="ThreadVis_234" context="dot_popup_234"/>
22     <tooltip orient="vertical" id="ThreadVis_234"/>
23     <popupset>
24         <popup id="dot_popup_234"/>
25     </popupset>
26     <box style="border-top: 1.59px solid rgb(0, 255, 255); ... top: ... opacity: 0.3; -moz-
           border-radius-topleft: 32px; -moz-border-radius-topright: 32px;"/>
27     <box style="top: 23.93px; left: ... -moz-border-radius-topleft: 9.57px; ..."/>
28     <box style="border: 1.59px solid black; top: ... -moz-border-radius-topleft: 12.76px;
           ..."/>
29     <box style="top: ..." tooltip="ThreadVis_258" context="dot_popup_258"/>
30     <tooltip orient="vertical" id="ThreadVis_258"/>
31     <popupset>
32         <popup id="dot_popup_258"/>
33     </popupset>
34     <box style="border-left: 1.59px solid rgb(255, 0, 127); ... top: ... -moz-border-radius-
           bottomleft: 32px; -moz-border-radius-bottomright: 32px;"/>
35     <box style="top: ... opacity: 0.3; -moz-border-radius-topleft: 9.57px; ..."/>
36     <box style="border: 1.59px solid black; top: ... -moz-border-radius-topleft: 12.76px;
           ..." hidden="true"/>
37     <box style="top: ..." tooltip="ThreadVis_282" context="dot_popup_282"/>
38     <tooltip orient="vertical" id="ThreadVis_282"/>
39     <popupset>
40         <popup id="dot_popup_282"/>
41     </popupset>
42     <box style="border-top: 1.59px solid rgb(153, 114, 0); ... top: ... opacity: 0.3; -moz-
           border-radius-topleft: 32px; -moz-border-radius-topright: 32px;"/>
43     <box style="width: 100%; height: 100%;" context="ThreadVisPopUp"/>
44     <description style="font-size: 9px; left: ..." value="1h" tooltiptext=" 1 hour 39
           minutes"/>
45     <description style="font-size: 9px; left: ..." value="4m" tooltiptext=" 4 minutes"
           hidden="true"/>
46     <description style="font-size: 9px; left: ..." value="2m" tooltiptext=" 2 minutes"
           hidden="true"/>
47     <description style="font-size: 9px; left: ..." value="6m" tooltiptext=" 6 minutes"
           hidden="true"/>
48 </stack>
```

Listing 8.2: An example visualisation using XUL (heavily simplified).

```
loop over all messages in conversation
    for each message, calculate time difference t to next message
    find minimal time difference tMin

calculate needed horizontal spacing
    sNeeded = 0
    loop over all messages in conversation
        for each message, set spacing s = t / tMin
        sNeeded += s

// maxWidth is the maximum available width for the visualisation
// sMin is the minimal spacing between two messages
sAvailable = maxWidth / sMin

while (sNeeded > sAvailable)
    sNeeded = 0
    loop over all messages in conversation
        for each message, set s = 0.9 * s, but minimally to 1
        sNeeded += s
    if (sNeeded == number of messages - 1)
        abort
```

Listing 8.3: The Time Scaling algorithm (in pseudo code).

minimal time difference between two message (`minimalTimeDifference`) and the available width for the visualisation (`width`).

8.4 Caching

As already discussed in Section 7.7, ThreadVis implements a caching algorithm to avoid unnecessary re-analysis of the same messages. The cache is automatically updated as new messages arrive in the user's mailbox. The implementation of the cache update is fairly simple: the extension stores a cache timestamp for each account in the mail client. When new messages arrive, the extension triggers a search across all enabled folders in the account. The search query is constructed using the last cache update timestamp and returns all mails that arrived after the last update. Only these messages are then analysed, the cache is updated and the new update timestamp is stored.

Section 8.1 already discussed the threading algorithm and how it ties into the cache information stored with each message. As each cache update only consists of a small number of messages, ThreadVis scales well to large mailboxes as the overall size and the total number of emails do not affect the calculation time, except for the first start, when all messages need to be analysed.

Listing 8.5 shows the algorithm used to determine the messages needed for a cache update. The `searchSession` object provides the required search methods. The `searchTerm` defines the query used in the search, in this case a date comparison (the attribute is `Date`, the search operator is `isAfter`). Then, a search listener is registered on the `searchSession` object and the search is executed (`searchSession.search(null)`). On every search hit (`onSearchHit`), the message is added to the threader. When the search is done (`onSearchDone`), all messages added to the threader are threaded and the cache information for all new messages is updated. After the cache is updated, the new update timestamp is written.

In case the search does not include the currently selected message (and thus the message lacks thread information), the search is repeated. The update timestamp is set to 10 minutes before the current mes-

```
1  Visualisation.prototype.timeScaling = function(containers,
2      minimalTimeDifference, width) {
3
4      var prefSpacing = THREADVIS.preferences.getPreference(
5          THREADVIS.preferences.PREF_VIS_SPACING);
6      var prefTimescaling = THREADVIS.preferences.getPreference(
7          THREADVIS.preferences.PREF_TIMESCALING);
8
9      // calculate the overall scale factor
10     var totalTimeScale = 0;
11     for (var counter = 0; counter < containers.length - 1; counter++) {
12         var thisContainer = containers[counter];
13         // norm the scale factor to the minimal time
14         thisContainer.xScaled = thisContainer.timeDifference
15             / minimalTimeDifference;
16         if (thisContainer.xScaled < 1) {
17             thisContainer.xScaled = 1;
18         }
19         totalTimeScale += thisContainer.xScaled;
20     }
21
22     var maxCountX = (width / prefSpacing) - 1;
23
24     var scaling = 0.9;
25     while (totalTimeScale > maxCountX) {
26         totalTimeScale = 0;
27         for (var counter = 0; counter < containers.length - 1; counter++)
               {
28             var thisContainer = containers[counter];
29             thisContainer.xScaled = thisContainer.xScaled * scaling;
30             if (thisContainer.xScaled < 1) {
31                 thisContainer.xScaled = 1;
32             }
33             totalTimeScale += thisContainer.xScaled;
34         }
35         if (totalTimeScale == containers.length - 1) {
36             break;
37         }
38     }
39     return containers;
40 }
```

Listing 8.4: The Time Scaling algorithm (in JavaScript, shortened).

```
1  Cache.prototype.updateNewMessagesInternal = function(message, doVisualise, accountKey,
       rootFolder) {
2    var searchSession = Components.classes["@mozilla.org/messenger/searchSession;1"]
3      .createInstance(Components.interfaces.nsIMsgSearchSession);
4    var searchTerm = searchSession.createTerm();
5    searchTerm.attrib = Components.interfaces.nsMsgSearchAttrib.Date;
6    searchTerm.op = Components.interfaces.nsMsgSearchOp.IsAfter;
7    var termValue = searchTerm.value;
8    termValue.attrib = searchTerm.attrib;
9    // get last update timestamp from preferences
10   var updateTimestamp = this.getLastUpdateTimestamp(accountKey);
11   termValue.date = updateTimestamp;
12   searchTerm.value = termValue;
13   searchSession.appendTerm(searchTerm);
14   this.addSubFolders(searchSession, rootFolder);
15   var newUpdateTimestamp = (new Date()).getTime() * 1000;
16   var ref = this;
17   var count = 0;
18   searchSession.registerListener({
19     onNewSearch: function() {
20       ref.cacheBuildCount++;
21     },
22     onSearchDone: function(status) {
23       ref.setLastUpdateTimestamp(accountKey, newUpdateTimestamp);
24       ref.threadvis.threader.thread();
25       var container = ref.threadvis.getThreader().findContainer(message.messageId);
26       if (container) {
27         if (container.isDummy()) {
28           var messageUpdateTimestamp = message.date - 1000000*60*10;
29           if (ref.cacheBuildCount > 1) {
30             messageUpdateTimestamp = 0;
31           }
32           ref.setLastUpdateTimestamp(accountKey, messageUpdateTimestamp);
33           ref.updateNewMessagesInternal(message, doVisualise, accountKey, rootFolder);
34           return;
35         }
36         ref.updateNewMessagesWriteCache(rootFolder, function() {
37           if (doVisualise) {
38             ref.threadvis.visualiseMessage(message);
39           }
40         });
41         ref.cacheBuildCount = 0;
42       } else {
43         var messageUpdateTimestamp = message.date - 1000000 * 60;
44         if (ref.cacheBuildCount > 1) {
45           messageUpdateTimestamp = 0;
46         }
47         if (ref.cacheBuildCount > 2) {
48           ref.cacheBuildCount = 0;
49           return;
50         }
51         ref.setLastUpdateTimestamp(accountKey, messageUpdateTimestamp);
52         ref.updateNewMessagesInternal(message, doVisualise, accountKey, rootFolder);
53       }
54     },
55     onSearchHit: function(header, folder) {
56       ref.threadvis.addMessage(header);
57       ref.getCacheInternal(header, true);
58       count++;
59       ref.newMessages.push(header);
60     }
61   });
62   searchSession.search(null);
63 }
```

Listing 8.5: Finding new messages to update the cache (in JavaScript, shortened).

sage to include it in the search. If the second search also fails to include the message, the update timestamp is reset to zero and the cache information for all messages in the account is updated. If this search on all messages in the account fails to find the current message, an error is displayed to the user and the update in cancelled.

8.5 Conclusion

Implementing an extension for any Mozilla-based product (like Firefox or Thunderbird) is very similar to writing a web-based application. Adding user interface elements using XUL very much resembles writing HTML code for web-pages. The styling of elements borrows ideas from current web standards and uses cascading style sheets (CSS). Interactivity is provided using JavaScript, again common to modern Web 2.0 applications. By using an interpreted language like JavaScript, no platform dependent implementation is needed and the extension can be installed on any Thunderbird installation.

The threading code used in the ThreadVis extension borrows from the built-in threading algorithm and extends it to work across different folders, which was necessary for ThreadVis to work. The visualisation itself uses a simple box model, which is styled using CSS. An implementation using SVG would be possible and even preferable, but unfortunately Thunderbird currently lacks the ability to render SVG.

The implementation of a caching architecture was necessary, as real-time threading is only possible up to a size of approximately 5000 documents. Using pre-calculated and cached information, the extension scales well to any number of messages, as only new messages need to be analysed. The initial threading, however, can take a long time, especially for larger mailboxes.

Chapter 9

Usage Study

"first impressions? - wow! I like it!..."

[ThreadVis feedback]

In contrast to the subjective "thinking aloud" test (see next chapter), an objective usage study was also conducted. In such an observational study, the typical use of a system is recorded for several users over a longer time period. The purpose is to find out which features of the system are used or unused. The resulting data is statistically analysed.

Section 9.1 describes the logging feature and the log file structure in detail. The log file uses a simple XML syntax, a built-in timer reminded users to send in the log files every three days. An evaluation of the collected log files is presented in Section 9.2. Both statistical data about the usage of ThreadVis and statistical data about the user's mailbox (number of messages, thread distribution) is discussed. Section 9.3 presents selected feedback from test users.

9.1 ThreadVis Log Files

ThreadVis contains a logging feature which collects anonymous usage data and saves it to a log file. In the initial start wizard, users are asked to enable the logging feature. Certain actions in the user interface (such as selecting a message, zooming, opening a pop-up window) trigger log file entries. Statistical information about the threaded messages (see Section 8.1 for details about the threading algorithm) is also included in the log file.

Listing 9.1 shows an example of a log file. On each start of the mail client, a new log file section is created. In each section, the currently installed version of the add-on is noted. Each log file entry contains a short description of the action, the date and time of the entry, and optional additional information.

To allow for easier debugging, additional information can be included in the log file (see Section 7.3.7). Users were reminded to send in the log files every three days. A wizard automatically composes a new email message and attaches the log file. After the message has been sent, the log file can be deleted with the click of a button.

9.2 Evaluation

A total of 25 users took part in the usage study and sent in the created log files over a period of up to three months. In those three months, the ThreadVis extension was downloaded over 230 times from the Mozilla Thunderbird add-on website [ThreadVis, 2008], which indicates a participation ratio of roughly 10%.

```
<log extensionversion="0.9.462">

<logitem date="Mon Apr 21 2008 19:11:20 GMT+0200" item="threadvis">
  <info key="action">startup</info>
</logitem>
<logitem date="Mon Apr 21 2008 19:11:21 GMT+0200" item="Using XUL."></
    logitem>
<logitem date="Mon Apr 21 2008 19:17:27 GMT+0200" item="msgselect">
  <info key="from">user</info>
  <info key="key">1468</info>
</logitem>
<logitem date="Mon Apr 21 2008 19:17:28 GMT+0200" item="threader">
  <info key="action">start</info>
</logitem>
<logitem date="Mon Apr 21 2008 19:17:28 GMT+0200" item="threader">
  <info key="action">end</info>
  <info key="totalMessages">4</info>
  <info key="totalThreads">1</info>
  <info key="msgPerThread">4</info>
  <info key="distribution">
    <info key="3">1</info>
  </info>
</logitem>
<logitem date="Mon Apr 21 2008 19:17:28 GMT+0200" item="visualise">
  <info key="msgkey">1468</info>
  <info key="top container">1466</info>
  <info key="msgcount">3</info>
</logitem>
 ...

</log>
```

Listing 9.1: An example of a ThreadVis usage log file.

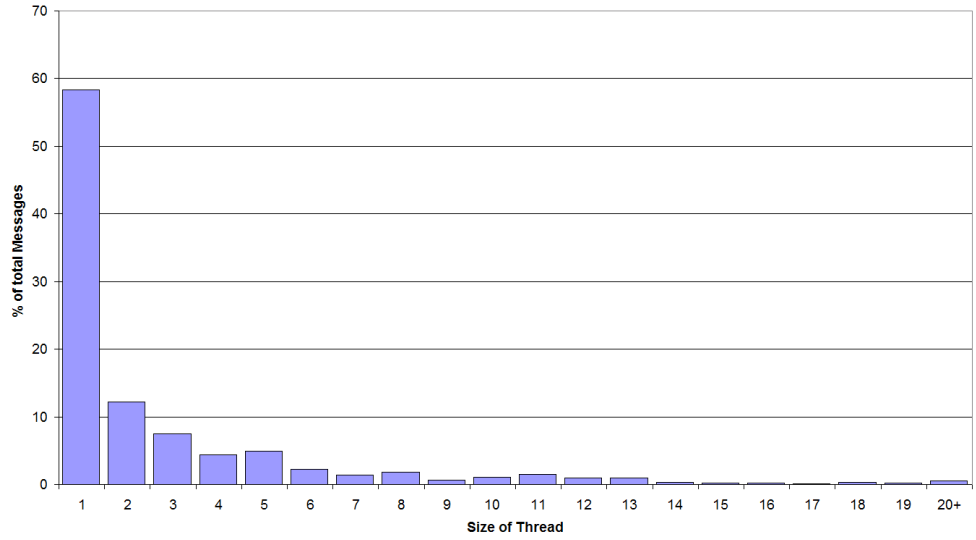


Figure 9.1: Thread size distribution of conversations visualised by the ThreadVis add-on.

9.2.1 Thread Size Distribution

For each visualised message, the log file includes the number of messages in the thread. This information was used to calculate the thread size distribution depicted in Figure 9.1. The absolute numbers are shown in Table 9.1. These findings strongly correlate to the result of the study described in Kerr [2003a]. 58% of all threads consisted of only a single message, the next most common thread size was 2. Over 90% of all conversations contained less than 7 messages.

9.2.2 Legend Window

Especially for larger threads or threads with many different contributors, the legend window provides additional information. Analysis of the log files showed that only 8 out of 25 users actually used the feature. This analysis shows that the legend window was mostly used for larger threads of size 10 and larger.

9.2.3 Pop-Up

ThreadVis also provides the possibility to open the visualisation in a larger, separate window. This larger pop-up is especially useful for very large threads, commonly found in newsgroup discussions. The data collected during the usage study shows that users either used the pop-up for very small threads (one to five messages) or for larger threads (15 messages and more).

9.2.4 Message Selection

ThreadVis not only acts as a passive visualisation of the conversation structure, but also as a means of navigating those messages. Comparing message selection in the normal message list and message selection using the ThreadVis visualisation shows that roughly 3% of all navigation events emanated from ThreadVis.

Comparing navigation events in the add-on to the size of the displayed conversation shows that the navigation feature is mostly used for small threads. Figure 9.2 shows the percentage of navigation events in ThreadVis as a function of thread size. Table 9.2 lists the total number of navigation events as a function of thread size.

Thread Size	*Number of Messages*
1	12623
2	2637
3	1619
4	956
5	1057
6	490
7	310
8	405
9	146
10	224
11	314
12	199
13	205
14	80
15	57
16	50
17	28
18	65
19	54
20+	112

Table 9.1: Thread size distribution of conversations visualised by ThreadVis.

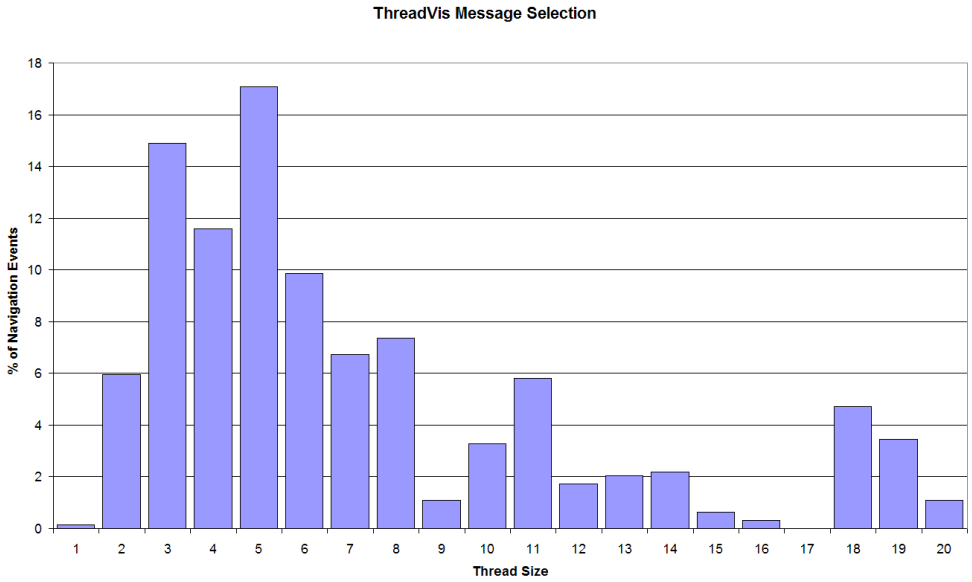


Figure 9.2: Usage of ThreadVis to select messages as a function of thread size.

Thread Size	*Number of Navigation Events*
1	1
2	38
3	95
4	74
5	109
6	63
7	43
8	47
9	7
10	21
11	37
12	11
13	13
14	14
15	4
16	2
17	0
18	30
19	22
20+	7

Table 9.2: Usage of ThreadVis to select messages as a function of thread size.

9.3 Selected Quotes

"I like to sort my mail by sender, thus I get messages scattered over several folders... but with your addon, I can trace up/down the thread just by clicking the "spots" on the visual..."

"Funktioniert! vielen Dank! Es erkennt sogar mails, die ich mit anderer Absender-Adressen gesendet habe als meins!"

"Die Thread-Ansicht mit klickbare Punkte ist einsame Spitze."

"Thank you for the excellent extension. Apparently, I can get one new adopeter per day merely running it in the office ..."

"nettes Programm... wie kam ich bisher nur ohne aus?"

"Anyway, thanks a lot for this extension, it's just great!"

9.4 Feedback

In addition to sending in the log files, many users also reported errors and even suggested new features or provided their own ideas. One particular user was extremely helpful and sent in over 20 suggestions, some of which have already been included in ThreadVis:

- When detecting sent mails, not only compare email address of current account but consider addresses of all accounts.

- Always use the same colour for sent messages. This makes the distinction between sent and received messages easier. (Sent mails already have a different form - hollow circle - than received mails.)
- The faded sub-threads are almost invisible on certain monitors. Increase the default opacity.
- Under certain conditions, the log-files contained private data such as email addresses.
- Make the colours for the different authors a user setting.
- Keep the size of the visualisation constant when navigating through a thread.
- A second click on the legend button should close the legend again.
- The date in the tooltip of a message is too long.
- Better handling of large mailboxes: automatically detect idle time and delay analysis.
- Rounding of time values in the timeline (for example, one day and 19 hours should be displayed as two days).
- Add a function to delete the entire thread.

9.5 Conclusion

By adding a logging feature to the ThreadVis add-on, anonymous usage data about the interaction of users with the add-on could be collected. Users were reminded by the add-on to send in the log files every three days. Many users not only sent in the log file data but provided feedback, reported errors and even suggested new features to implement.

Analysis of the collected data showed that the thread size distribution of visualised threads strongly correlates to the thread sizes found by Kerr [2003a]. Interesting to note is that the add-on accounts for over 3% of all message selection events. That means for every 30 clicks in the normal list of messages, one message was selected using the visualisation.

Chapter 10

Usability Study

> *"80% of maintenance is due to unmet or unforeseen user requirements; only 20% is due to bugs or reliability problems."*
>
> [Pressman [1986]]

To analyse the usability of the ThreadVis Mozilla Thunderbird extension, a thinking aloud test with six participants was conducted. The test setup assumes that novice users are installing the extension and using it without any prior training. The users were asked to perform routine tasks in the mail client. To follow their trail of thoughts during the test, the users were asked to "think aloud" while working. The test itself was done in German, as all test users were German native speakers, but the remarks and results are presented here in English.

First, in Sections 10.1 to 10.4, the test methodology is explained. A short overview over the test users is provided, as well as a description of the tasks and the test environment. Sections 10.5 to 10.9 discuss the results of the thinking aloud test and list positive findings and recommendations for future versions of the ThreadVis extension. The discussion also includes the results of the interviews conducted after the test and an evaluation of the feedback questionnaire.

10.1 Test Procedure

The test was conducted as a "thinking aloud" test in which test users are asked to perform certain tasks and to "think aloud" while they work. By asking the users to tell what they are trying to do, the decisions they make and any questions that come up, much information can be collected with relatively few users. A thinking aloud test can find many usability problems and why they occur, but having to think aloud can change the way users are solving problems, as it may force them to think before acting. Additionally, commenting their actions slows them down, making performance measures impossible. Table 10.1 gives an overview of the six test users who took part in the test. The target user group of the extension are regular email-users who like experimenting with new features and would install the extension in their own mail client.

10.2 Tasks

Each test user was asked to perform six different tasks (listed in Table 10.2). At first, the user was asked to spend a few minutes and play around with the mail client. After five minutes, the facilitator posed two questions: Whether the user noticed something in the email client and what was, in the user's opinion, the

Test User	*TP1 (Pilot)*	*TP2*	*TP3*	*TP4*	*TP5*	*TP6*
Date of Test	26. 4. 2008	27. 4. 2008	28. 4. 2008	27. 4. 2008	28. 4. 2008	28. 4. 2008
Time of Test	19:00	17:40	20:10	22:15	19:30	20:15
General Information						
First Name	Marlene	Sebastian	Oliver	Verena	Herbert	Manuel
Sex	female	male	male	female	male	male
Age	25	26	26	25	23	27
Education	Degree, Theology	Degree, BWL	Degree, BWL	Degree, Musician	A-levels	A-levels
Experience with Computers						
Using PC for	10 years	13 years	12 years	10 years	7 years	17 years
At PC (h/week)	20-25	25	50	15	30	70
Operating System	Windows	Windows	Windows	Windows	Windows	Windows, Unix
Experience with Internet and Web						
Online (h/week)	20	20	30-35	13	15-20	40
Online from	Home	Home	Office	Home	Home	Office
Internet Connection	DSL	DSL	DSL	DSL	DSL	DSL
Specific Questions						
Email-Client	GMail	Thunderbird, GMail	Outlook	Outlook	Thunderbird	Thunderbird, Outlook
Installed Add-ons	0	2	3	0	4	5
Experience with Usability Tests						
Participated in a user test previously	yes	no	no	no	no	no

Table 10.1: Overview of the six test users who participated in the thinking aloud test of ThreadVis.

meaning or purpose of the ThreadVis visualisation. For the following tasks, the user was given specific instructions, both verbally and printed, for later reference during the task.

The first, simple task acted as an ice-breaker, to get the user started and accustomed to the test methodology. The following, more complex tasks focused on the features of the ThreadVis extension. An additional task tested the usability of the settings dialogue for the extension. Users were given no prior training or explanation of the ThreadVis extension to test how intuitive the visualisation is to understand from scratch.

Task	*Description*	*Completion Criteria*	*Possible Solution Path*
1	First Impressions. Open the Mozilla Thunderbird mail client and spend a few minutes to look around.	User indicates they have finished looking around or 5 minutes have elapsed.	
2	Find the oldest message in the email conversation about "Top Keywords".	User has found the right conversation and has navigated to the first message.	Use the search box or scan through all mails. Find the first email in the conversation (which is a sent mail).
3	Find the newsgroup discussion about "Und noch eine knifflige GIS-Frage.." in the group at.gesellschaft.recht. How many messages does the discussion contain? How many persons participated in the conversation? How long did the conversation take place? Who wrote the most messages? How many answers did the fourth message receive? To that message was the last message an answer to?	User has selected a message in the conversation. Finds first and last email and counts participants. Analyses visualisation and finds answers to questions, for example using the legend window.	Search for the topic, analyse the ThreadVis visualisation to find the number of messages, navigate to first and last email to find the timeframe. Open the legend to count the number of participants.
4	The visualisation analyses the Junk-Mail folder, which is unnecessary. Disable the extension for that folder.	User has opened the options dialogue and deselected the Junk-Mail folder for the extension.	Right-click on the visualisation and choose "Settings" or open the options dialogue. Choose the ThreadVis tab and select "Disabled Accounts or Folders". Deselect the check box for the Junk-Mail folder.

Table 10.2: Test Tasks (Continued on Next Page...)

Computer Equipment	
Hardware	Fujitsu-Siemens Notebook, Intel Pentium 4, 1.8 GHz, 1024MB RAM
Operating System	Windows XP Professional
Mail-Client	Mozilla Thunderbird 2.0.0.12
ThreadVis	Version 0.9.462
Monitor Colours	16 bit
Monitor Resolution	1400 x 1050 set to 1024x786 for better visibility on video
Monitor Site	15" TFT
Video Equipment	
Camera	Panasonic NV-GS75 (miniDV)
Test Setting	
Location	Hasnerstraße 4/28, 1160 Vienna
Date of Pilot Text	26th April 2008
Date of Real Tests	27th and 28th April 2008

Table 10.3: The equipment and setting for the ThreadVis thinking aloud test.

Task	*Description*	*Completion Criteria*	*Possible Solution Path*
5	The newest newsgroup discussion in at.gesellschaft.recht discusses "email-disclaimer". Someone mentioned a website in that discussion? What is the URL of this website?	User names the URL.	User searches for the conversation, uses ThreadVis to navigate between messages in thread, finds correct message and names the URL.
6	Navigate to the email conversation about "top 5/10 keywords". The visualisation merged two discussions that do not belong together. Separate the messages into two distinct threads. Find the first message of the second thread (the first message that has nothing to do with the original thread) and remove it from the visualisation.	User right-clicks on the message and chooses "Remove message from thread".	Right-click on first message of new thread, choose "Remove message from thread".

Table 10.2: Test Tasks (Continued)

10.3 Test Environment

Table 10.3 lists the test equipment and program versions used for the test. The test was conducted on a local mailbox, so no internet connection was necessary. Each test was recorded on video for later analysis. To capture the facial expressions of the test users, a mirror was placed next to the monitor. Figure 10.1 shows a photograph of the test environment.

Figure 10.1: The usability test environment used for the ThreadVis thinking aloud test.

10.4 Interview Questions

After the user had completed all tasks, a short interview was conducted. The following questions were asked, as well as any additional questions that arose during the particular test:

1. How was it?
2. Did you intuitively understand the purpose of the visualisation or was it hard to understand?
3. Did you find the visualisation useful?
4. What would make the extension even more useful?
5. Would you install the extension yourself or recommend the extension to other users?

10.5 Discussion and Analysis

One of the main lessons learned from this thinking aloud test is that users need an introduction to the ThreadVis extension. The Thread Arcs visualisation itself was pretty self-explanatory for all users. The connection between the visualised messages and the normal mail client user interface on the other hand was not immediately visible. After users either discovered the connection themselves or were introduced to the visualisation, they generally found it useful and were able to complete the tasks using the extension. All users agreed on the visualisation not being distracting, with the negative effect that some users even overlooked the visualisation.

For two users, the Timeline captions displayed in the visualisation were distracting and led to confusion about what the abbreviated timespans meant (one user even associated the time difference of “2m” with a length measure of 2 metres). It was only after they discovered that the messages in the visualisation were ordered chronologically that those numbers must correlate to time differences.

Almost all users complained about the colour scheme used. Especially for larger conversations, the difference in colour for different authors becomes almost indistinguishable.

One user (TP4) did not find the zoom button next to the visualisation and had great trouble in even seeing the different messages as the conversation contained over 30 messages. The visualisation currently fits the whole visualisation in the available screen space. The default size can only be increased by a setting in the options dialogue.

The quick search feature of the mail client was used by almost all of the test users to find specific messages. As this search filters the displayed messages in the message list, the visualisation of the conversation can contain more messages (for example, if the subject changes) than the filtered list. Selecting such a message in the visualisation does not trigger a navigation event, as the message cannot be selected in the message list due to the applied filter.

As already mentioned before, the icons next to the visualisation (providing zoom, a pop-up view and the legend) were not perceived as "click"-able by many users. Even as the users discovered that these icons provided additional features, the functionality of each button was not clear by the icon itself and only the tooltip help text helped the users understand its purpose.

Greater emphasis must also be placed on the input possibilities of the visualisation. Users soon discovered that the visualisation displays the related messages to the currently displayed email, but did not recognise that they can use the visualisation to navigate through the thread. One test user (TP5) did not use the visualisation at all, the data collected from the feedback questionnaire is not taken into account when calculating the average and standard deviation.

10.6 Positive Findings

1. Visualisation does not Distract

None of the users was distracted by the visualisation. As the extension is using screen space that would otherwise be empty, the layout of other user interface elements is not changed. This confirms the goal of the extension to be an additional means to navigate messages, without changing the remaining functionality of the mail client.

2. Intuitive Visualisation

All users noted that the visualisation was easy to understand, once they understood the connection between the current conversation and the visualisation. Furthermore, the use of different colours for different authors was immediately obvious to the test users. That arcs connect related messages was also immediately obvious to the test users (see Figure 10.2).

3. Simple Setup

As soon as the users found the settings dialogue, it was very easy for them to find the right tab and disable the extension for the Junk-Mail folder. Since thread calculation is time-consuming, disabling the extension for unnecessary folders greatly improves the performance of the visualisation. Figure 10.3 shows a screen shot of the main options dialogue of the ThreadVis extension.

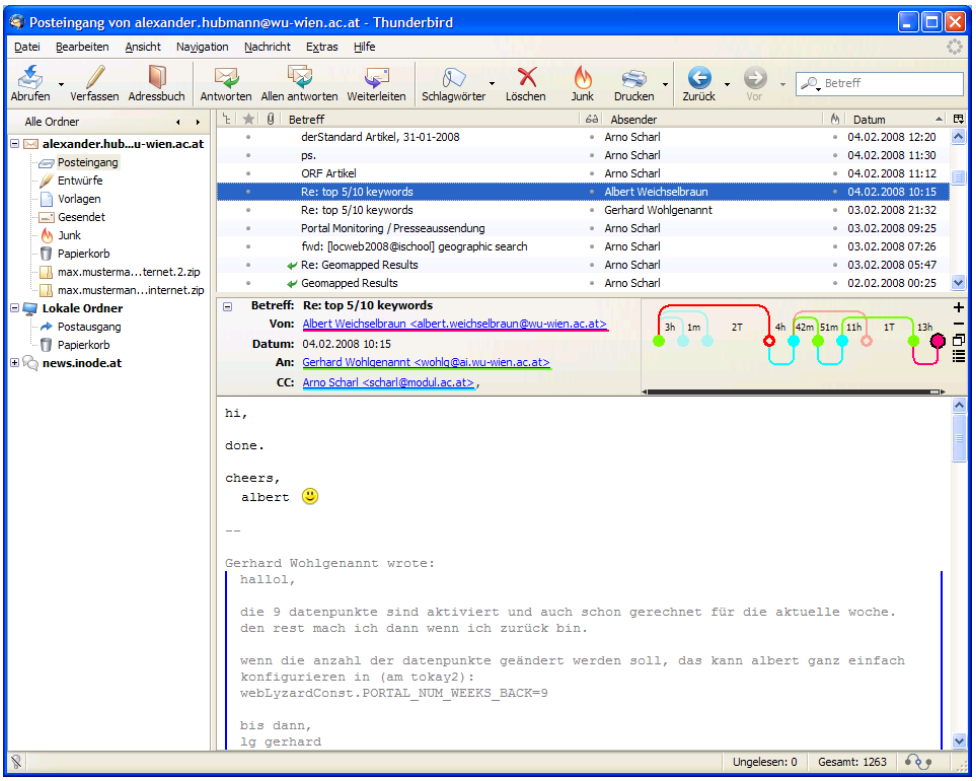


Figure 10.2: All users noted that the visualisation was easy to understand.

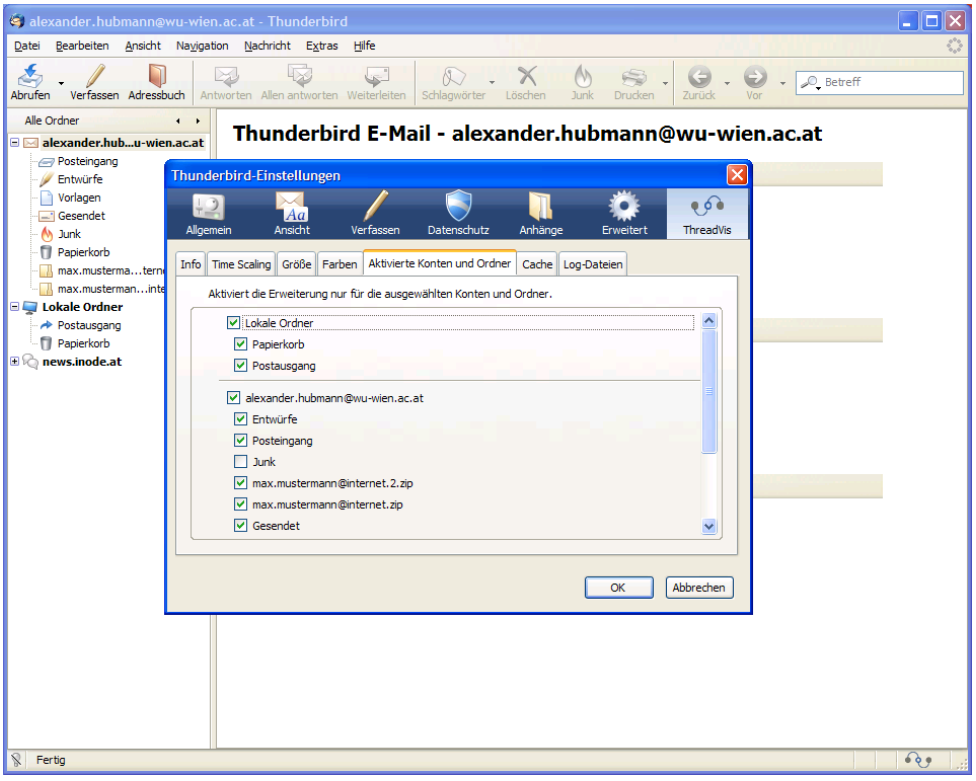


Figure 10.3: The ThreadVis Options dialogue to configure enabled accounts and folders.

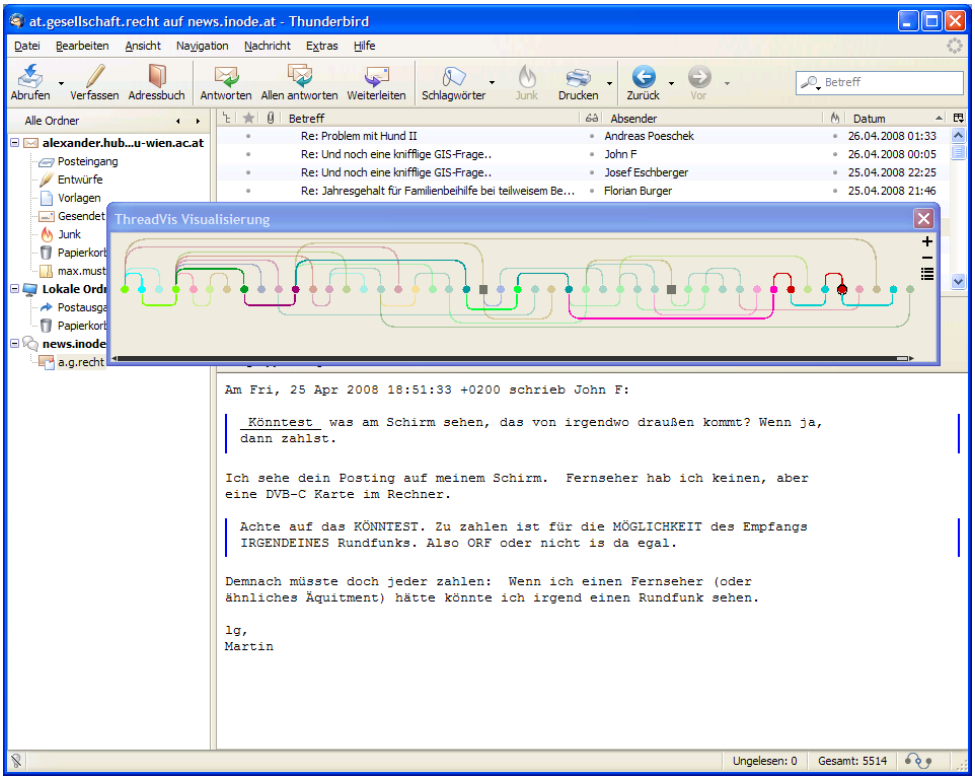


Figure 10.4: Participant colouring in ThreadVis: This conversation uses too many colours to be distinguishable.

10.7 List of Recommendations

1. Colours not distinguishable

TPs: TP1
Timestamps: 0:15:54 (TP1)

Figure 10.4 shows a screen shot of the conversation "Und noch eine knifflige GIS-Frage.." used in the thinking aloud test. In this case, the highlighting of authors by colours is more of a distraction than a help to the user. For larger threads, only the top 10 contributors could be assigned colours and the rest of the participants painted in grey. The user could also be given the option to select the authors to be highlighted, for example to focus on messages written by known authors.

2. Icons and Buttons

TPs: TP1, TP2, TP3
Timestamps: 0:18:04 (TP1), 0:53:09 (TP2), 0:55:10 (TP2), 1:02:36 (TP4)

Figure 10.5 shows the normal ThreadVis user interface. The buttons on the right provide zoom, a pop-up window, and the legend, but were not perceived as input buttons by the users. The functionality of the buttons was also not clearly conveyed by the icons. By making the buttons resemble normal user interface elements, more emphasis can be placed on the provided features. The icon set needs to be re-designed to better depict the purpose of the button.

Figure 10.5: The buttons in ThreadVis were not perceived as input buttons by the users. The icon set has to be re-designed as well, as the functionality of the buttons was not clearly conveyed by the icons.

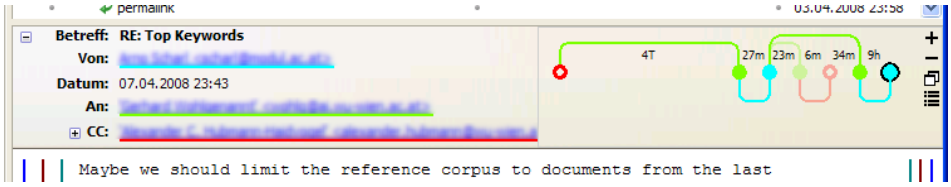


Figure 10.6: The timeline in ThreadVis. The labels displayed in the timeline were misinterpreted by the test users. Making the abbreviations non-ambiguous and avoiding unnecessary abbreviations would make the feature more intuitive.

3. Timeline Labels Ambiguous

TPs: TP1, TP3
Timestamps: 0:15:38 (TP1), 1:07:07 (TP3)

The labels in the displayed timeline (see Figure 10.6) were misinterpreted by the test users. One test user interpreted the "m" not as minutes, but as months, while another user even associated the caption "2m" with two metres instead of two minutes. It was also unclear whether the displayed time is measured since the beginning of the conversation or only between two adjacent messages.

More emphasis has to be placed on the chronology of the visualisation and the abbreviations have to be non-ambiguous. Space allowing, the time difference should not be abbreviated. In Figure 10.6, the first difference of four days does not have to be abbreviated and a label of "4 days" would be much more intuitive.

4. Quick Search and Visualisation

TPs: TP2, TP6
Timestamps: 1:04:25 (TP2), 1:00:00 (TP6)

Several users used the quick search feature in the mail client, which filters the current list view and only displays the matching elements. Figure 10.7 shows the ThreadVis visualisation for the selected conversation. It includes more messages than the current message listing shows. Clicking on a message which is excluded by the filter does not navigate to that message, which confuses the user. A possible solution would be to either display the message and disregard the applied filter or to remove the filter and display the whole message list again.

5. Default Zoom

TPs: TP1, TP2, TP4
Timestamps: 0:14:06 (TP1), 0:54:55 (TP2), 1:02:36 (TP4)

By default, the visualisation tries to fit the whole conversation into the available screen space. For large threads this leads to very small circles and arcs which makes the visualisation practically useless for navigating through the conversation, although it provides the user with an overview of the whole conversation.

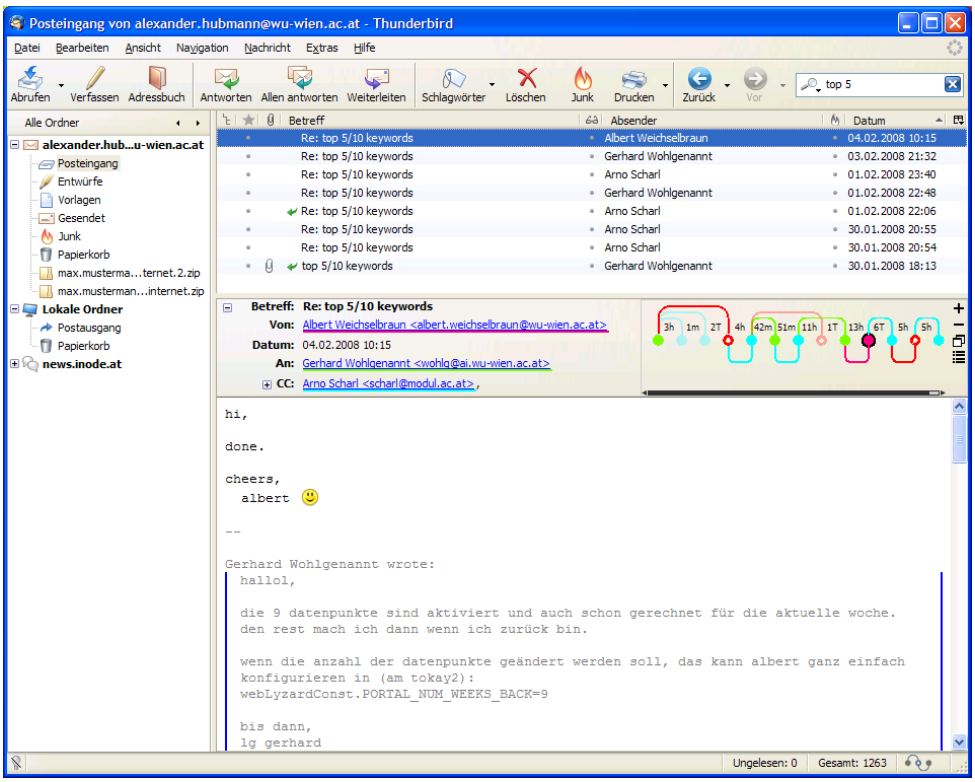


Figure 10.7: Quick search and ThreadVis. Using the quick search feature filters the message list, but ThreadVis still displays all messages in the conversation. Clicking on a filtered message does not navigate to that message, which confuses the user.

Figure 10.8: Default Zoom in ThreadVis. The visualisation tries to fit the whole conversation into the available screen space, which makes it practically useless for large threads. ThreadVis should only zoom out to an extent where details are still visible and display only a selected part of the visualisation.

The visualisation should only zoom out to an extent where details are still visible and provide a view of only a selected part of the conversation. Since ThreadVis automatically pans the visualisation when the user selects a new message, the displayed section automatically includes the currently displayed message.

10.8 Interviews

Most users complained about the colour scheme used in ThreadVis. For very large conversations, the colouring of authors is more a distraction than a help in distinguishing different authors. The users suggested using clearer colours and making the mapping between colours and authors more obvious. One user also complained about the pale colours (when unrelated sub-thread are faded out). This makes it even more difficult for the user to understand which author contributed which message as every author is then assigned two different colours.

The default zoom size was also criticised. The visualisation tries to fit the whole conversation into the available screen space which makes it practically useless for larger conversations. The users suggested

			TP1	*TP2*	*TP3*	*TP4*	*TP5*	*TP6*	*Av*	*Std Dev*
1. Visualisation distracts.										
Not at all	**6** 5 4 3 2 1 0	Definitely	6	6	6	5	6	6	5.80	0.45
2. Quality of the visualisation.										
Very good	6 5 **4** 3 2 1 0	Very poor	5	2	5	2	3	5	3.80	1.64
3. Visualisation is easy to understand.										
Very easy	6 **5** 4 3 2 1 0	Very hard	5	4	6	3	0	6	4.80	1.30
4. Appearance of the visualisation, including colours and graphics.										
Very good	6 5 4 **3** 2 1 0	Very poor	3	1	5	0	6	6	3.00	2.55
5. Speed of the extension.										
Very fast	6 **5** 4 3 2 1 0	Very slow	6	2	6	6	6	1	4.20	2.49
6. The extension is easy to configure.										
Very easy	6 5 **4** 3 2 1 0	Very difficult	3	6	6	4	3	3	4.40	1.52
7. The extension is relevant for me.										
Very relevant	6 5 4 3 **2** 1 0	Not relevant	4	0	4	0	1	3	2.20	2.05
8. Overall impression of the extension.										
Very good	6 **5** 4 3 2 1 0	Very poor	5	5	6	4	6	5	5.00	0.71
9. Will you install the extension yourself?										
Definitely	6 5 4 **3** 2 1 0	Never	5	6	0	3	6	3	3.40	2.31

Table 10.4: Responses to the feedback questionnaire. The neutral score in the original questionnaire has been converted into a weighted scale between 6 (best) and 0 (worst). Data collected from user 5 is not used in calculating the average and standard deviation as the user never actually used the visualisation.

only displaying a portion of the conversation and to zoom in by default.

The timeline feature was also a distraction for many users. One user noted that the Time Scaling visualisation already provides insight into the time difference between messages and that the timeline is not needed.

Two users also asked if the position of the arcs (top or bottom) has any significance. They were confused as the arcs were drawn seemingly random on top or on bottom.

For all users, the purpose of the visualisation was immediately obvious and they understood the extension intuitively. They especially commented on keeping an overview of the whole conversation. One user found it even more helpful after being told that the visualisation also works for messages stored in different folders, making it possible to keep messages sorted in folders and still have a thread overview.

When asked if they would install the extension themselves or recommend it to others, most users said that they would like to try out the extension, although they had doubts whether the visualisation would be useful for them.

10.9 Feedback Questionnaires

Table 10.4 shows the responses to the feedback questionnaire. The neutral scale in the original feedback questionnaire (see Section A.3) has been converted into a weighted scale between 6 (best) and 0 (worst).

Each user's response is included, as well as the calculated average and standard deviation. The calculated average is also marked on the given scale. Data collected from TP5 is not taken into account when calculating the average and standard deviation as the user never actually used the visualisation.

10.10 Conclusion

Since most users would install the extension from the Mozilla Thunderbird add-on page without any prior training, the thinking aloud test focused on untrained users to test the usability of the visualisation and the extension on users without any prior knowledge of the purpose of the visualisation. A short introduction to the general idea of the visualisation would have helped the users in completing the tasks more quickly, but the test without training is probably more realistic. Nevertheless, the test showed weak points of the extension and revealed certain usability issues.

To give novice users a better start at using the extension, the wizard displayed when the user first starts the mail client after the extension is installed needs to be enhanced. The general idea of the visualisation and possible interactions of the user with the extension need to be explained in detail.

The Time Scaling was well perceived by the users, the timeline was perceived as rather distracting. The labels placed on the timeline need better abbreviations as the currently used letters are too unclear.

The used colour scheme was criticised by all test users. For small threads, the colours are useful and the user can connect the used colour to the author. For larger threads, the different colours were mostly perceived as a distraction and degraded the usability. One idea would be to limit the number of colours and give the user the option to only highlight certain participants in the conversation, assigning a default grey colour to the remaining authors. That way, the user can focus on contributions by certain authors.

The various input features of the visualisation such as navigating through the conversation, zooming and displaying the legend have to be presented more prominently to the user. Buttons placed next to the visualisation were not perceived as buttons, as they lacked the common look-and-feel of normal buttons in the user interface.

In the subsequent interviews almost every user pointed out that the visualisation was interesting and provided a good overview of the whole conversation. Although the personal relevance of the extension was rated fairly low (2 out of 6 points), three users indicated that they would definitely install the extension themselves. Altogether, the users rated the overall impression extension very high (5 out of 6 points), even though it still needs to improve on its usability.

Chapter 11

Outlook

"The best thing about the future is that it comes only one day at a time."

[Abraham Lincoln]

The ThreadVis Thunderbird extension has matured over the course of its development, with fewer and fewer errors and issues being reported by users. Nevertheless, several open issues, possible enhancements, and new ideas remain.

In its current implementation, ThreadVis uses simple DOM elements to draw the visualisation elements. Although anti-aliasing slightly improves the image quality of the visualisation, the use of CSS styles is less than ideal. Scalable Vector Graphics (SVG) would be a prime candidate to be used to draw the visualisation. Current versions of Mozilla Thunderbird do not support SVG elements in the user interface, but as soon as such versions become available, the extension could be easily adapted.

As the extension leverages existing web-techniques such as DOM manipulation, CSS styles and most importantly, JavaScript, it could be ported to web browsers to be included in web mail clients. ThreadVis could either be included in an open source web mail platform such as the Horde Project [Horde, 2008] or be transformed into a Mozilla Firefox extension to enhance existing, even closed-source, web mail interfaces such as Google Mail or Windows Live Hotmail. Google Mail already provides the user with a conversation-centric interface listing all messages belonging to a thread alongside the current message. As all messages in a conversation are already present in the interface, only the visualisation part has to be ported. Additionally, Google is adding more and more APIs to their products, making integration even easier.

Even using the caching techniques discussed earlier, very large mailboxes (some of the testers had inboxes larger than six gigabytes) still pose serious problems for the extension. On the first start, when no cache information is available, all messages in the user's mailbox need to be analysed. For very large numbers of messages, the first analysis could be done in several smaller chunks in the background. As soon as the whole cache has been built, the user is notified and the visualisation is displayed. Building an intelligent queuing system that prioritises messages and folders the user currently views would provide a ThreadVis visualisation as soon as possible while delaying analysis of unrelated messages.

In addition to colour-coding contributors in the visualisation and underlining their names in the header view, also the quotes in the email body could be marked using the same colours. Currently, Thunderbird assigns colours to quoted text based on the nesting level. By using the same colours to underline the author's name, to visualise the author's messages in the ThreadVis visualisation, and to highlight the author's quotes in the displayed email, users could immediately link the participants in the conversation to the text.

Chapter 12

Concluding Remarks

"To everything there is a season. A time for every purpose under heaven."

[Koh 3:1 NKJ]

This thesis presented the ThreadVis extension for Mozilla Thunderbird. It enhances the Thread Arcs visualisation presented in Chapter 5 and makes it available in a widely used email client. First, an overview of various information visualisation techniques was given in Chapter 2. Chapter 3 discussed a selection of common email clients, both for desktop use and web mail access. A more detailed look at visualisations targeted at email clients was presented in Chapter 4. The Thread Arcs visualisation, on which the ThreadVis extension is based, was described in detail in Chapter 5. Chapter 6 introduced extension development for Mozilla Thunderbird. The features of the ThreadVis extension were discussed in Chapter 7. Chapter 8 presented selected details of the implementation. A usage study of ThreadVis was conducted, where users were asked to send in anonymous usage logs. The results and findings of the study were discussed in Chapter 9. A thinking aloud test of ThreadVis was presented in Chapter 10, including a discussion of insights and recommendations for ThreadVis. Finally, Chapter 11 discussed possible future enhancements and future work on the ThreadVis extension.

Appendix A

Original Materials

This appendix includes the original materials used in the thinking aloud test of ThreadVis. The thinking aloud test was conducted in German, since all of the test users were German native speakers.

A.1 Vertraulichkeits- und Einverständniserklärung

Danke, dass Sie an meiner Studie teilnehmen. Bitte beachten Sie, dass Ihnen unter Umständen vertrauliche Informationen zuteil werden und dass Sie diese nicht weitergeben dürfen. Bild- und Tonaufnahmen werden von Ihrer Sitzung gemacht, um es anderen, die heute nicht anwesend sein können, zu ermöglichen, aus Ihrem Feedback Nutzen zu ziehen.

Bitte lesen Sie die untenstehende Einverständniserklärung und unterschreiben Sie an der dafür vorgesehenen Stelle. Vielen Dank.

Ich erkläre, keine Informationen aus der Studie weiterzugeben.

Ich weiß, dass Bild- und Tonaufnahmen von meiner Sitzung gemacht werden. Ich gebe die Erlaubnis, diese Aufnahmen für Lehrzwecke und im Rahmen wissenschaftlicher Forschung zu verwenden.

Ort:

Datum:

Name:

Geburtsdatum:

Unterschrift:

A.2 Hintergrundbefragung

Danke dass Sie sich als Freiwilliger für meinen Test zur Verfügung stellen.
Bitte beantworten Sie die folgenden Fragen:

1. Angaben zur Person

Geschlecht: [ ] männlich [ ] weiblich
Alter: _____
Beruf: _____

2. Sehvermögen

1. Verwenden Sie eine Sehhilfe bei der Arbeit am Computer?
   [ ] Keine [ ] Kontaktlinsen
   [ ] Brille [ ] Sonstige: _____

2. Sind Sie farbenblind?
   [ ] Nein [ ] Ja, und zwar _____

3. Ausbildung

1. Abgeschlossene Ausbildung:
   [ ] Lehre [ ] Studium
   [ ] Matura [ ] Doktorat

2. Wenn Sie studieren oder studiert haben, beschreiben Sie bitte Ihr Hauptstudiengebiet:
   _____

4. Umgang mit Computern

1. Wie lange benutzen Sie bereits Personal Computer?
   ____ Jahre

2. Wie viele Stunden pro Woche verwenden Sie einen Computer?
   ____ Stunden

3. Welche Art von Computer / Betriebssystem verwenden Sie am meisten?
   [ ] Microsoft Windows [ ] Unix
   [ ] Apple Macintosh [ ] Sonstige: _____

5.Umgang mit dem Internet und Web

1. Wie viele Stunden pro Woche benutzen Sie das Internet?
   ____ Stunden

2. Von wo aus surfen Sie am meisten?
   [ ] Zuhause [ ] Uni
   [ ] Büro [ ] Sonstiges

Table A.1: Background questionnaire given before the test. (Continued on Next Page. . .)

3. Welche Art von Internet Zugang verwenden Sie normalerweise?
   [ ] Analog Modem [ ] ADSL/xDSL
   [ ] ISDN Modem [ ] Sonstige: _____
   [ ] Telekabel (Chello)

6. Spezielle Fragen

1. Welches E-Mail-Programm verwenden Sie normalerweise?
   [ ] Mozilla Thunderbird [ ] Webmail: _____
   [ ] Microsoft Outlook [ ] Sonstige: _____

2. Falls Sie Mozilla Firefox / Thunderbird verwenden, haben Sie schon Add-ons installiert?
   [ ] Nein [ ] Ja, und zwar schon ca. ____ Stück

3. Sagt Ihnen der Begriff “Information Visualisation” etwas?
   [ ] Nichts [ ] Ein wenig [ ] Kenne ich

7. Erfahrung mit Usability Tests

1. Haben Sie schon an eine Usability Studie teilgenommen?
   [ ] als Testperson
   [ ] als Mitglied des Testteams
   Wenn ja, was war das für eine Studie?
   _____

Table A.1: Background questionnaire given before the test. (Continued)

A.3 Feedback Formular

1.	Die Visualisierung lenkt ab.	Auf jeden Fall	3	2	1	0	1	2	3	Überhaupt nicht
2.	Qualität der Visualisierung.	Sehr gut	3	2	1	0	1	2	3	Sehr schlecht
3.	Die Visualisierung ist einfach zu verstehen.	Sehr einfach	3	2	1	0	1	2	3	Sehr kompliziert
4.	Grafische Gestaltung der Visualisierung, inklusive Farben und Grafiken.	Sehr gut	3	2	1	0	1	2	3	Sehr schlecht
5.	Geschwindigkeit der Visualisierung	Sehr schnell	3	2	1	0	1	2	3	Sehr langsam
6.	Die Visualisierung ist einfach zu konfigurieren.	Sehr einfach	3	2	1	0	1	2	3	Sehr kompliziert
7.	Die Visualisierung ist relevant für mich.	Sehr relevant	3	2	1	0	1	2	3	Überhaupt nicht
8.	Gesamteindruck der Visualisierung.	Sehr gut	3	2	1	0	1	2	3	Sehr schlecht
9.	Werden Sie die Visualisierung selbst installieren?	Auf jeden Fall	3	2	1	0	1	2	3	Sicher nicht
	Positive Eindrücke: Negative Eindrücke:									

Table A.2: The feedback questionnaire given after the test (Feedback Formular).

A.4 Test Tasks

1. Erste Eindrücke.
   Öffne den Mozilla Thunderbird Mail Client und sieh' dich ein paar Minuten um.

2. Finde die älteste Nachricht in der E-Mail Unterhaltung zum Thema "Top Keywords".

3. Finde die Newsgroup-Diskussion (in der Gruppe at.gesellschaft.recht) zum Thema
   "Und noch eine knifflige GIS-Frage..".
   Wie viele Beiträge umfasst die Diskussion insgesamt?
   Wie viele Personen waren an der Diskussion beteiligt?
   Über welchen Zeitraum hat sich die Diskussion erstreckt?
   Wer hat die meisten Nachrichten zur Diskussion beigetragen?
   Wie viele Antworten hat die 4. Nachricht bekommen?
   Auf welche Nachricht war die letzte Nachricht eine Antwort?

4. Die Visualisierung analysiert unnützerweise den Junk-Mail-Ordner.
   Deaktiviere die Visualisierung für den Junk-Mail-Ordner.

5. In der neuesten Newsgroup-Diskussion zum Thema "email-disclaimer" in at.gesellschaft.recht erwähnte jemand eine Webseite.
   Wie ist die Adresse dieser Seite?

6. Navigiere zur E-Mail-Diskussion zum Thema "top 5/10 keywords".
   Die Visualisierung hat hier fälschlicherweise zwei verschiedene Diskussionen zu einer zusammengezogen.
   Finde die erste Nachricht des zweiten Threads (die erste Nachricht, die nichts mehr mit der ursprünglichen Diskussion zu tun hat) und entferne sie aus der Visualisierung.

A.5 Orientation Script

Hallo, mein Name ist Sascha. Ich bin hier, um eine Erweiterung eines E-Mail-Programmes zu testen und möchte dich dabei um deine Hilfe bitten.

Ich werde dich auffordern, einige typische Abläufe durchzuführen. Versuche dein Ihr Bestes, aber mach dir keine Gedanken über deinen Erfolg, die Erweiterung wird getestet und nicht dein Können. Da die Erweiterung nicht perfekt ist, kann es sicher noch da und dort ecken, kleinere Fehler können auftreten und nicht alles wird so funktionieren, wie man es sich erwartet.

Meine einzige Rolle heute ist, die Vor- und Nachteile des Designs von dir aus gesehen aufzunehmen. Für mich ist es wichtig, deine echte, ehrliche Meinung zu bekommen.

Um auch während des Tests deinen Gedanken folgen zu können, bitten ich dich “Laut Mitzudenken”, das heißt deine Handlungen und Eindrücke mitzukommentieren. Stelle auftretende Fragen bitte zu jeder Zeit, allerdings darf ich diese wahrscheinlich erst am Ende des Tests beantworten.

Während du arbeitest, werde ich einige Notizen machen. Für jene, die heute nicht hier sein können, werden wir die Tests auch auf Video aufnehmen.

Wenn du dich unwohl fühlst, kannst du den Test jederzeit abbrechen.

Hast du Fragen?

Wenn nicht, lass uns damit beginnen, eine kurze Hintergrundbefragung auszufüllen.

A.6 Test Transcripts

This section lists the chronological protocols of all conducted thinking aloud tests: Table A.3 for TP1, Marlene; Table A.4 for TP2, Sebastian; Table A.5 for TP3, Oliver; Table A.6 for TP4, Verena; Table A.7 for TP5, Herbert and Table A.8 for TP6, Manuel.

Time hh:mm:ss	Event
00:00:02	Enters room.
00:00:14	Orientation script.
00:01:20	Consent form.
00:02:07	Background questionnaire.
00:04:27	Demo of thinking aloud test.
00:05:20	Start task 1.
00:07:00	Opens new email dialogue.
00:07:35	In offline mode, only send later is available.
00:08:27	End task 1.
00:08:28	Start task 2.
00:09:21	Searches with quotes, does not find message.
00:09:50	User wants to do a full text search.
00:10:20	Search did not return the expected result.
00:11:21	User discovers quick search feature.
00:12:02	User found wrong discussion.
	Facilitator note: Need to redo task / prepare mails better.
00:12:30	End task 2.
00:12:55	Start task 3.
00:13:43	Uses quick search.
00:14:06	Uses small ThreadVis visualisation.
00:14:15	Opens the legend window.
00:14:44	Discovers connection between colours and authors.
00:14:50	Opens a larger pop-up visualisation.
00:15:02	Counts messages in visualisation.
00:15:30	Unknown messages.
00:15:38	Misinterpreted timeline. Read “11m” as 11 months instead of 11 minutes.
00:15:54	Colours difficult to distinguish, too pale.
00:16:26	Discovers visualisation is also input element to navigate.
00:16:28	Double clicks message, new window opens.
00:16:43	Opens legend.
00:17:24	End task 3.
00:17:27	Start task 4.
00:17:45	Right clicks on visualisation. Context menu for message does not contain link to settings.
00:18:04	Notes that a general ThreadVis button would be a good idea.
00:18:53	Opens properties dialogue for folder, cannot configure visualisation here.
00:21:00	Discovers add-on menu entry.
00:21:08	Opens settings dialogue for add-on.
00:21:20	End task 4.
00:21:50	Start task 5.
00:23:13	End task 5.
00:23:27	Start task 6.
00:24:20	Discovers that there are more messages in the visualisation than in the list.
00:28:15	Finds wrong message.
00:29:05	Right clicks message, uses context menu to remove message.
00:29:07	End task 6.
00:29:20	Begin interview.
00:37:00	End interview.
00:37:20	Feedback questionnaire.
00:38:05	End.

Table A.3: TP1: Marlene. Times on first video tape.

Time hh:mm:ss	Event
00:38:05	Start test.
00:38:18	Orientation script.
00:39:30	Consent form.
00:40:30	Background questionnaire.
00:44:00	Demo thinking aloud test.
00:45:07	Start task 1.
00:46:10	Opens add-on dialogue too see which add-ons are installed.
00:46:53	End task 1.
00:46:55	Start task 2.
00:47:12	Searches form messages.
00:47:46	Sorts result by date.
00:47:58	End task 2.
00:48:00	Start task 3.
00:49:10	Uses search again.
00:50:50	Refines search.
00:51:55	Test facilitator gives hint to visualisation.
00:52:45	“Many coloured lines.”
00:53:09	Button symbols, uses tooltip to discover functionality.
00:53:15	Opens legend window.
00:53:30	Wants to click on author in legend window.
00:54:10	Total number of messages not visible.
00:54:55	Visualisation is too small, cannot see details.
00:55:10	Zooms visualisation but reads tooltip for button first.
00:55:42	Arcs between messages not clear.
00:56:03	Understands visualisation.
00:57:50	End task 3.
00:58:00	Start task 4.
00:58:15	Right clicks on visualisation, chooses settings.
00:58:40	End task 4.
00:58:47	Start task 5.
01:02:30	End task 5.
01:02:40	Start task 6.
01:04:25	Tries to click on message in visualisation.
	Quick search prohibits navigation.
01:04:55	End task 6.
01:05:07	Begin interview.
01:11:45	End interview.
01:11:48	Feedback questionnaire.
01:15:15	End.

Table A.4: TP2: Sebastian. Times on first video tape.

Time hh:mm:ss	Event
00:00:00	Start test.
00:00:14	Orientation script.
00:01:17	Consent form.
00:02:02	Background questionnaire.
00:04:18	Demo of thinking aloud test.
00:05:22	Start task 1.
00:06:55	“What are newsgroups?”
00:10:25	End task 1.
00:10:27	Start task 2.
00:11:17	Discovers quick search.
00:12:00	Repeats search in sent-mail folder.
00:12:28	End task 2.
00:12:34	Start task 3.
00:13:20	Uses quick search.
00:16:50	Discovers visualisation.
00:17:07	Visualisation is a little bit confusing.
00:17:13	Opens pop-up window.
00:17:40	Understands arcs immediately, counts replies.
00:18:44	End task 3.
00:18:46	Start task 4.
00:19:20	Tries to open settings dialogue by right clicking on folder.
00:19:50	Opens menu entry for settings dialogue.
00:20:05	Disables add-on for all folders.
00:20:30	Discovers tab for activated accounts and folders.
00:20:50	End task 4.
00:21:02	Start task 5.
00:22:20	Tries combined search.
00:24:50	End task 5.
00:25:05	Start task 6.
00:25:55	Opens email, opens po-pup.
00:26:39	Subject of messages does not start with “Re” in tooltip.
00:27:24	Help by text facilitator that subject changes.
00:29:20	Test user does not understand that visualisation also displays sent mails.
00:29:35	Clicks on sent mail, jumps to sent-mail folder.
00:30:38	Right clicks on message and removes it from the visualisation.
00:30:44	End task 6.
00:30:50	Begin interview.
00:38:10	Feedback questionnaire.
00:42:00	End.

Table A.5: TP3: Oliver. Times on second video tape.

Time hh:mm:ss	Event
00:42:08	Start test.
00:42:12	Orientation script.
00:43:20	Consent form.
00:44:25	Background questionnaire.
00:47:05	Demo thinking aloud test.
00:48:00	Start task 1
00:51:00	“What are keywords?”
00:52:06	End task 1.
00:52:10	Start task 2.
00:52:28	Sorts by subject.
00:54:30	Only looks in inbox, knows that the original message has to be there.
	Does not search in the sent-mail folder.
00:56:10	End task 2.
00:56:30	Start task 3.
00:57:43	Sorts by subject, searches for message.
01:00:38	Discovers visualisation, tries to understand colours.
01:01:07	Understands visualisation displays message relationships.
01:01:26	Discovers that one colour is mapped to one author. Counts colours in small visualisation.
01:02:36	Complains that visualisation is so small. Overlooks zoom buttons.
01:03:10	Authors are underlined in colour.
01:06:58	Discovers zoom button.
01:07:07	Misinterprets timeline labels, reads “11m” as 11 metres.
01:07:30	Tries to make sense of timeline labels. Thinks that “m” stands for messages.
01:08:18	Understands purpose of visualisation, “who sent which message to whom”.
01:08:48	Opens legend.
01:10:05	Arcs unclear, does it visualise six answers or six persons?
01:12:18	End task 3.
01:12:24	Start task 4.
01:13:08	Opens folder properties.
01:13:45	Tries right clicking on visualisation.
01:14:32	Test facilitator asks if task is really done. (User disabled visualisation for all folders.)
01:15:55	End task 4.
01:16:05	Start task 5.
01:18:30	End task 5.
01:18:37	Start task 6.
01:19:58	Right clicks on message.
01:21:02	End task 6.
01.21:13	Begin interview.
01:25:15	End interview.
01:15:20	Feedback questionnaire.
01:27:26	End.

Table A.6: TP4: Verena. Times on second video tape.

Time hh:mm:ss	Event
00:00:01	Start test.
00:00:09	Orientation script.
00:01:16	Consent form.
00:02:17	Background questionnaire.
00:06:15	Demo thinking aloud test.
00:07:20	Start task 1.
00:10:10	End task 1.
00:10:19	Start task 2.
00:10:55	Facilitator gives advice to read task again.
00:12:25	End task 2.
00:12:30	Start task 3.
00:15:07	Facilitator gives advice to read task again.
00:21:46	End task 3.
00:21:55	Start task 4.
00:24:25	End task 4.
00:24:30	Start task 5.
00:26:25	End task 5.
00:26:31	Start task 6.
00:33:05	End task 6.
00:33:10	Begin interview.
00:35.55	End interview.
00:36:00	Feedback questionnaire.
00:41:14	End.

Table A.7: TP5: Herbert. Times on third video tape.

Time hh:mm:ss	Event
00:41:15	Start test.
00:41:25	Orientation script.
00:42:30	Consent form.
00:43:35	Background questionnaire.
00:45:20	Demo thinking aloud test.
00:46:33	Start task 1.
00:47:18	Opens add-on dialogue, discovers installed add-on.
00:48:13	Opens settings dialogue for the add-on.
00:49:24	End task 1.
00:49:30	Start task 2.
00:50:26	End task 2.
00:50:31	Start task 3.
00:52:20	Sorts and groups messages.
00:53:48	Changes to threaded view in message list.
00:54:57	Thread list is unclear, not visible who replied to whom.
00:55:55	Facilitator gives hint about visualisation.
00:56:11	Understand visualisation, “messages on timeline”.
00:56:43	Finds right answer using visualisation.
00:57:00	End task 3.
00:57:04	Start task 4.
00:57:17	Already discovered add-on menu, opens settings dialogue.
00:57:30	End task 4.
00:57:45	Start task 5.
00:58:50	End task 5.
00:58:55	Start task 6.
01:00:00	Visualisation ignores search term.
01:02:00	Right clicks on message and removes it from the visualisation.
01:02:10	End task 6.
01:02:15	Begin interview.
01:07:05	End interview.
01:07:10	Feedback questionnaire.
01:08:00	End.

Table A.8: TP6: Manuel. Times on third video tape.

Bibliography

Keith Andrews [2002a]. *Visual Exploration of Large Hierarchies with Information Pyramids*. In *Proc. 6th International Conference on Information Visualisation (InfoVis 2002)*, pages 793–798. ISSN 10939547. doi:10.1109/IV.2002.1028871. (Cited on page 15.)

Keith Andrews [2002b]. *Visualising Information Structures: Aspects of Information Visualisation*. Professorial Thesis, Graz University of Technology. ftp://ftp.iicm.edu/pub/keith/habil/visinfo.pdf. (Cited on pages 3, 16 and 17.)

Keith Andrews [2006]. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Graz University of Technology, Austria. http://ftp.iicm.edu/pub/keith/thesis/. (Cited on page xv.)

Keith Andrews and Helmut Heidegger [1998]. *Information Slices: Visualising and Exploring Large Hierarchies using Cascading, Semi-Circular Discs*. In *Late Breaking Hot Topics Paper, IEEE Symposium on Information Visualization (InfoVis '98)*, pages 9–12. http://www.iicm.tu-graz.ac.at/liberation/iicm_papers/ivis98.pdf. (Cited on pages 10 and 11.)

Keith Andrews, Michael Pichler, and Peter Wolf [1996]. *Towards Rich Information Landscapes for Visualising Structured Web Spaces*. In *Proc. 2nd IEEE Symposium on Information Visualisation (InfoVis '96)*, pages 62–63. doi:10.1109/INFVIS.1996.559218. (Cited on page 15.)

Keith Andrews, Josef Wolte, and Michael Pichler [1997]. *Information Pyramids: A New Approach to Visualising Large Hierarchies*. In *Late Breaking Hot Topics Paper, IEEE Symposium on Visualization (Vis '97)*, pages 49–52. http://www.iicm.tu-graz.ac.at/liberation/iicm_papers/vis97.pdf. (Cited on pages 15 and 16.)

Keith Andrews, Christian Gütl, Josef Moser, Vedran Sabol, and Wilfried Lackner [2001]. *Search Result Visualisation with xFIND*. In *Proc. 2nd International Workshop on User Interfaces to Data Intensive Systems (UIDIS 2001)*, pages 50–58. doi:10.1109/UIDIS.2001.929925. (Cited on pages 20 and 22.)

Apple [2008]. *Apple Cocoa Homepage*. http://developer.apple.com/cocoa/. [Online; accessed 01-September-2008]. (Cited on page 26.)

Thomas Ball and Stephen G. Eick [1996]. *Software Visualization in the Large*. IEEE Computer, 29(4), pages 33–43. doi:10.1109/2.488299. http://citeseer.ist.psu.edu/ball96software.html. (Cited on pages 4 and 6.)

John Bandhauer [2000]. *Scriptable Components (XPConnect)*. http://www.mozilla.org/scriptable/. [Online; accessed 01-September-2008]. (Cited on page 64.)

Richard Boardman [2000]. *Bubble Trees: The Visualization of Hierarchical Information Structures*. In *CHI 2000 Extended Abstracts on Human Factors in Computing Systems (CHI 2000)*, pages 315–316. ACM Press, New York, NY, USA. ISBN 1581132484. doi:10.1145/633292.633484. (Cited on pages xv, 11 and 13.)

BusinessObjects [2008]. *TableLens SDK*. http://www.businessobjects.com/product/catalog/tablelens/. [Online; accessed 01-September-2008]. (Cited on page 18.)

Phil Dykstra [1991]. *XDU*. http://sd.wareonearth.com/~phil/xdu/. [Online; accessed 01-September-2008]. (Cited on page 13.)

Stephen C. Eick, Joseph L. Steffen, and Eric E. Sumner Jr. [1992]. *Seesoft - A Tool for Visualizing Line Oriented Software Statistics*. IEEE Transactions on Software Engineering, 18(11), pages 957–968. doi:10.1109/32.177365. (Cited on page 4.)

Scott Fertig, Eric Freeman, and David Gelernter [1996]. *Lifestreams: An Alternative to the Desktop Metaphor*. Conference Companion on Human Factors in Computing Systems (CHI '96), pages 410–411. doi:10.1145/257089.257404. (Cited on pages 6 and 7.)

Danyel Fisher and Paul Moody [2001]. *Studies of Automated Collection of Email Records*. University of Irvine, Technical Report, UCI-ISR-02-4. http://www.isr.uci.edu/tech_reports/UCI-ISR-02-4.pdf. (Cited on pages 43, 44 and 47.)

Eric T. Freeman and Scott J. Fertig [1995]. *Lifestreams: Organizing Your Electronic Life*. In *AAAI Fall Symposium: AI Applications in Knowledge Navigation and Retrieval*. (Cited on page 6.)

G. W. Furnas [1986]. *Generalized Fisheye Views*. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '86)*, pages 16–23. ACM Press, New York, NY, USA. ISBN 0897911806. doi:10.1145/22627.22342. (Cited on page 4.)

GNOME [2008a]. *GNOME Evolution Homepage*. http://www.gnome.org/projects/evolution/. [Online; accessed 01-September-2008]. (Cited on pages 25 and 27.)

GNOME [2008b]. *GNOME Homepage*. http://www.gnome.org. [Online; accessed 01-September-2008]. (Cited on page 25.)

GNUMail.app [2006]. *GNUMail.app Homepage*. http://www.collaboration-world.com/gnumail/. [Online; accessed 01-September-2008]. (Cited on pages 26 and 27.)

GNUStep [2008]. *GNUStep Homepage*. http://www.gnustep.org. [Online; accessed 01-September-2008]. (Cited on page 26.)

Amit Goel [2006]. *Parallel Coordinates Visualization Applet*. http://www.amitgoel.com/projects/parallel-coordinates/. [Online; accessed 01-September-2008]. (Cited on page 19.)

Google [2006]. *Gmail Homepage*. http://www.gmail.com/. [Online; accessed 01-September-2008]. (Cited on page 26.)

Daniel Gruen, Steven L. Rohall, Suzanne Minassian, Bernard Kerr, Paul Moody, Bob Stachel, Martin Wattenberg, and Eric Wilcox [2004]. *Lessons from the ReMail Prototypes*. In *Proc. 2004 ACM Conference on Computer Supported Cooperative Work (CSCW 2004)*, pages 152–161. ACM Press, New York, NY, USA. ISBN 1581138105. doi:10.1145/1031607.1031634. (Cited on page 31.)

Christian Gütl [2000]. *xFIND: Extended Framework for Information Discovery*. Doctoral Thesis, Graz University of Technology. http://www.iicm.tugraz.at/thesis/cguetl_diss/diss_cguetl.pdf. (Cited on page 20.)

Marti Hearst [2003]. *Information Visualization: Principles, Promise, and Pragmatics*. Tutorial at CHI 2003 Conference on Human Factors in Computing Systems (CHI 2003). http://bailando.sims.berkeley.edu/talks/chi03-tutorial.ppt. (Cited on page 3.)

Beth Hetzler, W. Michelle Harris, Susan Havre, and Paul Whitney [1998]. *Visualizing the Full Spectrum of Document Relationships*. In *Proc. Fifth International Society for Knowledge Organization (ISKO) Conference*, pages 168–175. http://infoviz.pnl.gov/pdf/isko.pdf. (Cited on pages 20 and 21.)

Horde [2008]. *IMP Webmail Client*. http://www.horde.org/imp/. [Online; accessed 01-September-2008]. (Cited on page 113.)

InfoScope [2008]. *InfoScope Homepage*. http://www.macrofocus.com/public/products/infoscope/. [Online; accessed 01-September-2008]. (Cited on page 17.)

Alfred Inselberg and Bernard Dimsdale [1990]. *Parallel Coordinates: A Tool for Visualizing Multi-Dimensional Geometry*. In *Proc. 1st Conference on Visualization 1990 (Vis '90)*, pages 361–378. IEEE Computer Society Press, Los Alamitos, CA, USA. ISBN 0818620838. (Cited on page 17.)

Inxight [2006]. *Inxight TimeWall*. http://www.inxight.com/products/sdks/tw/. Inxight is now part of Business Objects, the presented Java applet is no longer available. (Cited on pages 4 and 5.)

ISO/IEC [2004]. *9834-8:2004 Information Technology, "Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components"*. IEEE Computer Graphics and Applications. (Cited on page 62.)

Dean F. Jerding and John T. Stasko [1998]. *The Information Mural: A Technique for Displaying and Navigating Large Information Spaces*. IEEE Transactions on Visualisation and Computer Graphics, 4(3), pages 257–271. doi:10.1109/2945.722299. (Cited on page 4.)

Brian Johnson and Ben Shneiderman [1991]. *Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures*. In *Proc. 1991 IEEE Conference on Visualization (Vis '91)*, pages 284–291. San Diago, California, USA. doi:10.1109/VISUAL.1991.175815. (Cited on page 12.)

Bernard Kerr [2003a]. *Thread Arcs: An Email Thread Visualization*. In *IEEE Symposium on Information Visualization (InfoVis 2003)*, pages 211–218. IEEE. doi:10.1109/INFVIS.2003.1249028. http://www.research.ibm.com/remail/threadarcs.html. (Cited on pages xv, 31, 43, 44, 45, 46, 47, 49, 73, 80, 81, 83, 88, 97 and 100.)

Bernard Kerr [2003b]. *IBM Research Report. Thread Arcs: An Email Thread Visualization*. http://domino.watson.ibm.com/library/cyberdig.nsf/1e4115aea78b6e7c85256b360066f0d4/7a30ed0aac59bf5d85256d79006f272f?OpenDocument. [Online; accessed 01-September-2008]. (Cited on pages 43, 46 and 48.)

Bernard Kerr and Eric Wilcox [2004]. *Designing Remail: Reinventing the Email Client Through Innovation and Integration*. In *CHI 2004 Extended Abstracts on Human Factors in Computing Systems (CHI 2004)*, pages 837–852. ACM Press, New York, NY, USA. ISBN 1581137036. doi:10.1145/985921.985944. (Cited on pages 31 and 32.)

John Lamping and Ramana Rao [1994]. *Laying Out and Visualizing Large Trees Using a Hyperbolic Space*. In *Proc. 7th Annual ACM Symposium on User Interface Software and Technology (UIST 1994)*, pages 13–14. ACM Press, New York, NY, USA. ISBN 0897916573. doi:10.1145/192426.192430. (Cited on page 8.)

John Lamping, Ramana Rao, and Peter Pirolli [1995]. *A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies*. In *Proc. SIGCHI Conference on Human Factors in*

Computing Systems (CHI '95), pages 401–408. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. ISBN 0201847051. doi:10.1145/223904.223956. (Cited on pages 8 and 10.)

Anton Leuski [2006]. *eArchivarius Homepage*. http://people.ict.usc.edu/~leuski/projects/earchivarius/index.html. [Online; accessed 01-September-2008]. (Cited on page 33.)

Anton Leuski, Douglas W. Oard, and Rahul Bhagat [2003]. *eArchivarius: Accessing Collections of Electronic Mail*. In *Proc. 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval (SIGIR 2003)*, pages 468–468. ACM Press, New York, NY, USA. ISBN 1581136463. doi:10.1145/860435.860559. (Cited on page 33.)

Wei-Jen Li, Shlomo Hershkop, and Salvatore J. Stolfo [2004]. *Email Archive Analysis Through Graphical Visualization*. In *Proc. 2004 ACM Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEV 2004)*, pages 128–132. ACM Press, New York, NY, USA. ISBN 1581139748. doi:10.1145/1029208.1029229. (Cited on pages xv, 35 and 36.)

Jock D. Mackinlay, George G. Robertson, and Stuart K. Card [1991]. *The Perspective Wall: Detail and Context Smoothly Integrated*. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*, pages 173–176. ACM Press, New York, NY, USA. ISBN 0897913833. doi:10.1145/108844.108870. (Cited on pages xv, 4 and 5.)

Mirko Mandic and Andruid Kerne [2004]. *faMailiar - Intimacy-Based Email Visualization*. In *Proc. IEEE Symposium on Information Visualization (InfoVis 2004)*, page 215.14. IEEE Computer Society, Washington, DC, USA. ISBN 0780387793. doi:10.1109/INFOVIS.2004.26. (Cited on page 36.)

Mirko Mandic and Andruid Kerne [2005]. *Using Intimacy, Chronology and Zooming to Visualize Rhythms in Email Experience*. In *CHI 2005 Extended Abstracts on Human Factors in Computing Systems (CHI 2005)*, pages 1617–1620. ACM Press, New York, NY, USA. ISBN 1595930027. doi:10.1145/1056808.1056980. (Cited on pages 36, 37 and 38.)

Dan Matejka [2005]. *Introduction to XUL*. http://developer.mozilla.org/en/Introduction_to_XUL. [Online; accessed 01-September-2008]. (Cited on page 51.)

Microsoft [2007a]. *Outlook*. http://www.microsoft.com/outlook/. [Online; accessed 01-September-2008]. (Cited on page 24.)

Microsoft [2007b]. *Outlook Express*. http://www.microsoft.com/windows/oe/. [Online; accessed 01-September-2008]. (Cited on page 23.)

Microsoft [2008a]. *Microsoft Live Hotmail Screen Shot Gallery*. http://www.microsoft.com/presspass/gallery/screenshots/windowslive.mspx. [Online; accessed 01-September-2008]. (Cited on page 29.)

Microsoft [2008b]. *Microsoft Office Screen Shot Gallery*. http://www.microsoft.com/presspass/gallery/screenshots/office.mspx. [Online; accessed 01-September-2008]. (Cited on page 24.)

Mozilla [1998]. *XPIDL syntax*. http://www.mozilla.org/scriptable/xpidl/syntax.html. [Online; accessed 01-September-2008]. (Cited on page 62.)

Mozilla [2000]. *XPCOM Project Homepage*. http://www.mozilla.org/projects/xpcom/index.html. [Online; accessed 01-September-2008]. (Cited on page 62.)

Mozilla [2001]. *XUL Programmer's Reference Manual*. http://developer.mozilla.org/En/XUL_Reference. [Online; accessed 01-September-2008]. (Cited on page 52.)

Mozilla [2005]. *XUL Overlays*. http://developer.mozilla.org/en/XUL_Overlays. [Online; accessed 01-September-2008]. (Cited on page 57.)

Mozilla [2006a]. *Extension Versioning, Update and Compatibility*. http://developer.mozilla.org/en/Extension_Versioning,_Update_and_Compatibility. [Online; accessed 01-September-2008]. (Cited on page 69.)

Mozilla [2006b]. *Mozilla Project Homepage*. http://www.mozilla.org/. [Online; accessed 01-September-2008]. (Cited on page 24.)

Mozilla [2006c]. *Mozilla Homepage*. http://www.mozilla.com/. [Online; accessed 01-September-2008]. (Cited on page 24.)

Mozilla [2006d]. *Mozilla Localization Project*. http://www.mozilla.org/projects/l10n/mlp.html. [Online; accessed 01-September-2008]. (Cited on page 70.)

Mozilla [2006e]. *Toolkit Version Format*. http://developer.mozilla.org/en/Toolkit_version_format. [Online; accessed 01-September-2008]. (Cited on page 69.)

Mozilla [2006f]. *XBL*. http://developer.mozilla.org/en/XBL. [Online; accessed 01-September-2008]. (Cited on page 60.)

Mozilla [2006g]. *XBL 1.0 Reference*. http://developer.mozilla.org/En/XBL:XBL_1.0_Reference. [Online; accessed 01-September-2008]. (Cited on page 60.)

Mozilla [2006h]. *Cross-Platform Installer Module (XPI)*. http://developer.mozilla.org/en/XPI. [Online; accessed 01-September-2008]. (Cited on page 66.)

Mozilla [2006i]. *Install Manifests*. http://developer.mozilla.org/en/install.rdf. [Online; accessed 01-September-2008]. (Cited on page 67.)

Mozilla [2006j]. *XUL*. http://developer.mozilla.org/en/XUL. [Online; accessed 01-September-2008]. (Cited on page 52.)

Mozilla [2006k]. *XUL Tutorial*. http://developer.mozilla.org/en/XUL_Tutorial. [Online; accessed 01-September-2008]. (Cited on pages 52 and 53.)

Mozilla [2006l]. *XUL Tutorial: Localization*. http://developer.mozilla.org/En/XUL_Tutorial:Localization. [Online; accessed 01-September-2008]. (Cited on page 70.)

Mozilla [2006m]. *XUL Tutorial: Introduction to XBL*. http://developer.mozilla.org/En/XUL_Tutorial:Introduction_to_XBL. [Online; accessed 01-September-2008]. (Cited on pages 60 and 61.)

Mozilla [2008a]. *Chrome Registration*. http://developer.mozilla.org/en/Chrome_Registration. [Online; accessed 01-September-2008]. (Cited on page 60.)

Mozilla [2008b]. *Mozilla Add-On Portal*. http://addons.mozilla.org. [Online; accessed 01-September-2008]. (Cited on page 72.)

Mozilla [2008c]. *Core JavaScript 1.5 Guide*. http://developer.mozilla.org/en/Core_JavaScript_1.5_Guide. [Online; accessed 01-September-2008]. (Cited on page 66.)

Mozilla [2008d]. *Components Object*. http://developer.mozilla.org/en/Components_object. [Online; accessed 01-September-2008]. (Cited on page 64.)

David A. Nation [1998]. *WebTOC: A Tool to Visualize and Quantify Web Sites Using a Hierarchical Table of Contents Browser*. In *CHI 1998 Conference Summary on Human Factors in Computing Systems (CHI '98)*, pages 185–186. ACM Press, New York, NY, USA. ISBN 1581130287. doi:10.1145/286498.286664. (Cited on page 8.)

David A. Nation, Catherine Plaisant, Gary Marchionini, and Anita Komlodi [1997]. *Visualizing Websites Using a Hierarchical Table of Contents Browser: WebTOC*. In *Proc. 3rd Conference on Human Factors and the Web*. http://hcil.cs.umd.edu/trs/97-10/97-10.html. (Cited on pages 8 and 9.)

Rick Parrish [2001a]. *XPCOM Part 2: XPCOM Component Basics*. http://www-128.ibm.com/developerworks/webservices/library/co-xpcom2.html. [Online; accessed 01-September-2008]. (Cited on pages 62, 63 and 64.)

Rick Parrish [2001b]. *XPCOM Part 1: An Introduction to XPCOM*. http://www-128.ibm.com/developerworks/webservices/library/co-xpcom.html. [Online; accessed 01-September-2008]. (Cited on page 62.)

Pidgin [2008]. *Pidgin Instant Messenger Homepage*. http://www.pidgin.im. [Online; accessed 01-September-2008]. (Cited on page 25.)

Jon Postel [1980]. *DoD Standard Internet Protocol*. RFC 760. http://www.ietf.org/rfc/rfc760.txt. Obsoleted by RFC 791, updated by RFC 777. (Cited on page 85.)

R. S. Pressman [1986]. *Software Engineering: A Practitioner's Approach (2nd ed.)*. McGraw-Hill, Inc., New York, NY, USA. ISBN 0-070-50783-X. (Cited on page 101.)

KDE Project [2006]. *KMail Homepage*. http://kontact.kde.org/kmail/. [Online; accessed 01-September-2008]. (Cited on page 25.)

Ramana Rao and Stuart K. Card [1994]. *The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus + Context Visualization for Tabular Information*. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '94)*, pages 318–322. ACM Press, New York, NY, USA. ISBN 0897916506. doi:10.1145/191666.191776. (Cited on pages xv, 18 and 19.)

P. Resnick [2001]. *Internet Message Format*. RFC 2822 (Proposed Standard). http://www.ietf.org/rfc/rfc2822.txt. (Cited on page 85.)

George G. Robertson, Jock D. Mackinlay, and Stuart K. Card [1991]. *Cone Trees: Animated 3D Visualizations of Hierarchical Information*. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*, pages 189–194. ACM Press, New York, NY, USA. ISBN 0897913833. doi:10.1145/108844.108883. (Cited on pages xv, 13 and 14.)

Steven L. Rohall [2003]. *Interactive Poster: Visualizations in the ReMail Prototype*. In *IEEE Symposium on Information Visualization (InfoVis 2003)*. http://domino.research.ibm.com/cambridge/research.nsf/0/e82de19dfc1354b285256dd3007454c1/$FILE/TR2003-15.pdf. (Cited on page 31.)

Steven L. Rohall, Daniel Gruen, Paul Moody, and Seymour Kellerman [2001]. *Email Visualizations to Aid Communications*. In *Proc. IEEE Symposion on Information Visualization (InfoVis 2001)*, pages 12–15. (Cited on pages 31, 45 and 47.)

Steven L. Rohall, Dan Gruen, Paul Moody, Martin Wattenberg, Mia Stern, Bernard Kerr, Bob Stachel, Kushal Dave, Robert Armes, and Eric Wilcox [2004]. *ReMail: A Reinvented Email Prototype*. In *CHI 2004 Extended Abstracts on Human Factors in Computing Systems (CHI 2004)*, pages 791–792. ACM Press, New York, NY, USA. ISBN 1581137036. doi:10.1145/985921.985938. (Cited on page 31.)

Vedran Sabol [2001]. *Visualisation Islands: Interactive Visualisation and Clustering of Search Result Sets*. Master's Thesis, Graz University of Technology. ftp://ftp.iicm.edu/pub/thesis/vsabol.pdf. (Cited on page 20.)

Maryam Samiei, John Dill, and Arthur Kirkpatrick [2004]. *EzMail: Using Information Visualization Techniques to Help Manage Email*. In *Proc. Eighth International Conference on Information Visualisation (InfoVis 2004)*, pages 477–482. IEEE. doi:10.1109/IV.2004.1320187. (Cited on pages 33 and 34.)

Jürgen Schipflinger [1998]. *The Design and Implementation of the Harmony Session Manager*. Master's Thesis, Graz University of Technology. http://www.iicm.edu/thesis/jschipf.pdf. (Cited on page 15.)

Ben Schneiderman and Martin Wattenberg [2001]. *Ordered Treemap Layouts*. In *Proc. IEEE Symposium on Information Visualization (InfoVis 2001)*, pages 73–78. IEEE. ISSN 1522404X. doi:10.1109/INFVIS.2001.963283. (Cited on page 12.)

Ben Shneiderman [1992]. *Tree Visualization with Tree-Maps: 2-d Space-Filling Approach*. ACM Transactions on Graphics, 11(1), pages 92–99. ISSN 0730-0301. doi:10.1145/102377.115768. (Cited on page 12.)

Ben Shneiderman [1996]. *The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations*. In *IEEE Visual Languages*, UMCP-CSD CS-TR-3665, pages 336–343. College Park, Maryland 20742, U.S.A. doi:10.1109/VL.1996.545307. http://www.cs.ubc.ca/~tmm/courses/cs533c-02/readings/shneiderman96eyes.pdf. (Cited on page 3.)

SmartMoney [2006]. *Map of the Market*. http://www.smartmoney.com/map-of-the-market/. [Online; accessed 01-September-2008]. (Cited on page 14.)

Marc A. Smith and Andrew T. Fiore [2001]. *Visualization Components for Persistent Conversations*. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI 2001)*, pages 136–143. ACM Press, New York, NY, USA. ISBN 1581133278. doi:10.1145/365024.365073. (Cited on page 45.)

John Stasko and Eugene Zhang [2000]. *Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations*. In *Proc. IEEE Symposium on Information Visualization (InfoVis 2000)*, page 57. IEEE Computer Society, Washington, DC, USA. ISBN 0769508049. (Cited on pages 10 and 12.)

John Stasko, Richard Catrambone, Mark Guzdial, and Kevin McDonald [2000]. *An Evaluation of Space-Filling Information Visualizations for Depicting Hierarchical Structures*. International Journal of Human-Computer Studies, 53(5), pages 663–694. ISSN 10715819. doi:10.1006/ijhc.2000.0420. (Cited on page 10.)

Jim Thomas, Paula Cowley, Olga Kuchar, Lucy Nowell, Judi Thomson, and Pak Chung Wong [2001]. *Discovering Knowledge Through Visual Analysis*. Journal of Universal Computer Science, 7(6), pages 517–529. http://www.jucs.org/jucs_7_6/discovering_knowledge_through_visual. (Cited on pages 20 and 21.)

ThreadVis [2008]. *ThreadVis Extension Homepage*. https://addons.mozilla.org/de/thunderbird/addon/6533. [Online; accessed 01-September-2008]. (Cited on page 95.)

Doug Turner and Ian Oeschger [2003]. *Creating XPCOM Components*. Brownhen Publishing, First Edition. http://www.mozilla.org/projects/xpcom/book/cxc/pdf/cxc.pdf. (Cited on page 64.)

Lisa Tweedie, Bob Spence, David Williams, and Ravinder Bhogal [1994]. *The Attribute Explorer*. In *Conference Companion on Human Factors in Computing Systems (CHI '94)*, pages 435–436. ACM Press, New York, NY, USA. ISBN 0897916514. doi:10.1145/259963.260433. (Cited on pages xv, 18 and 20.)

UrhG [2008]. *Urheberrechtsgesetz §46*. http://www.ris2.bka.gv.at/Dokumente/Bundesnormen/NOR40041624/NOR40041624.html. [Online; accessed 01-September-2008]. (Cited on pages 6, 7, 8, 9, 10, 11, 12, 16, 17, 18, 21, 22, 32, 34, 37, 38, 46, 47 and 48.)

Gina Danielle Venolia and Carman Neustaedter [2003]. *Understanding Sequence and Reply Relationships within Email Conversations: A Mixed-Model Visualization*. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI 2003)*, pages 361–368. ACM Press, New York, NY, USA. ISBN 1581136307. doi:10.1145/642611.642674. (Cited on pages xv, 39, 40, 43 and 45.)

Fernanda B. Viégas, Scott Golder, and Judith Donath [2006]. *Visualizing Email Content: Portraying Relationships from Conversational Histories*. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI 2006)*, pages 979–988. ACM Press, New York, NY, USA. ISBN 1595933727. doi:10.1145/1124772.1124919. (Cited on pages xv, 37, 38 and 39.)

W3C [2006]. *Cascading Style Sheets*. http://www.w3.org/Style/CSS/. [Online; accessed 01-September-2008]. (Cited on page 86.)

W3C [2008a]. *CSS3 Backgrounds and Borders Module*. http://www.w3.org/TR/css3-border/#the-border-radius. [Online; accessed 01-September-2008]. (Cited on page 86.)

W3C [2008b]. *Document Object Model (DOM)*. http://www.w3.org/DOM/. [Online; accessed 01-September-2008]. (Cited on page 86.)

W3C [2008c]. *Scalable Vector Graphics (SVG)*. http://www.w3.org/Graphics/SVG/. [Online; accessed 01-September-2008]. (Cited on page 86.)

Martin Wattenberg [1999]. *Visualizing the Stock Market*. In *CHI 1999 Extended Abstracts on Human Factors in Computing Systems (CHI '99)*, pages 188–189. ACM Press, New York, NY, USA. ISBN 1581131585. doi:10.1145/632716.632834. (Cited on page 12.)

Martin Wattenberg [2002]. *Arc Diagrams: Visualizing Structure in Strings*. In *Proc. IEEE Symposium on Information Visualization (InfoVis 2002)*, page 110. IEEE Computer Society, Washington, DC, USA. ISBN 076951751X. doi:10.1109/INFVIS.2002.1173155. (Cited on pages 6, 7, 8 and 45.)

Wikipedia [2007]. *Microsoft Outlook — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/wiki/Microsoft_Outlook. [Online; accessed 18-February-2007]. (Cited on page 24.)

Wikipedia [2008a]. *Gmail — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/wiki/Gmail. [Online; accessed 15-April-2008]. (Cited on page 28.)

Wikipedia [2008b]. *History of Mozilla Application Suite — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/wiki/History_of_Mozilla_Application_Suite. [Online; accessed 15-April-2008]. (Cited on page 24.)

Wikipedia [2008c]. *Outlook Express — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/wiki/Outlook_Express. [Online; accessed 15-April-2008]. (Cited on page 23.)

Peter Wolf [1996]. *Three-Dimensional Information Visualisation: The Harmony Information Landscape*. Master's Thesis, Graz University of Technology. http://ftp.iicm.tugraz.at/pub/theses/pwolf.pdf. (Cited on page 15.)

Andrew M. Wood, Nick S. Drew, Russel Beale, and Bob J. Hendley [1995]. *HyperSpace: Web Browsing with Visualisation*. In *Proc. WWW3*, pages 21–25. http://www.igd.fhg.de/www/www95/proceedings/posters/35/index.html. (Cited on pages 17 and 18.)

XULPlanet [2006]. *XULPlanet Homepage*. http://www.xulplanet.com/. [Online; accessed 01-September-2008]. (Cited on page 52.)