Czech Technical University, Prague
Faculty of Electrical Engineering

Bachelor Thesis

Cascading undo control

Jan Vratislav

Supervisor: Ing. Ivo Malý

Study Program: Electronics and Computer Science Engineering

Study Branch: Computer Science and Engineering

June 2008

Declaration

I hereby declare that I have completed this bachelor thesis independently and that I have listed all the literature and publications used.

I have no objection to usage of this work in compliance with the act §60 Zakona c. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

Prague, June 12th, 2008 .....

Acknowledgement

I thank Chris Fernandes and Aaron Cass for the great cooperation and the guidance they have been providing me. I also thank to Lee L. Davenport'37 for the grant which helped me to work on this thesis.

I thank to Ing. Ivo Malý for accepting and supporting my thesis.

I thank to my family and my friends for their support.

Abstract

The so-called cascading undo command has been introduced by Aaron Cass and Chris Fernandes [CF06]. This new approach to the undo command overcomes the weakness of the linear undo command. It allows undoing an arbitrary action from the history while watching the dependencies among the actions. However, there is not a visualization of cascading undo yet. Thus, in this thesis we discuss, introduce, develop, and evaluate several visualizations for cascading undo. Unlike the linear undo visualizations, cascading undo visualizations have to deal with dependencies among user actions. We believe that an overview of the dependencies should be presented to a user before committing and undo command. The visualizations we proposed are flexible enough to reflect the possible complexity of the user actions and their dependencies.

Abstrakt

Aaron Cass a Chris Fernandes představili takzvaný kaskádový příkaz zpět [CF06]. Tento nový způsob příkazu zpět překonává slabiny všudypřítomného lineárního příkazu zpět. Umožňuje totiž zrušit libovolnou akci z historie akcí dokumentu. Při tom bere v potaz závislosti mezi těmito akcemi. Nicméně nikdo ještě nevyvinul vizualizaci pro kaskádový příkaz zpět. V této práci diskutujeme, představujeme, vyvíjíme a hodnotíme několik vizualizací kaskádového příkazu zpět. Na rozdíl od vizualizace lineárního příkazu zpět musí kaskádový příkaz zpět počítat se vzájemnými závislostmi provedených akcí. Věříme, že uživatelé by měli mít přehled o těchto závislostech ještě před jejich vlastním odebráním. Námi navržené vizualizace jsou flexibilní natolik, aby zvládly zobrazit komplexitu uživatelských akcí a závislostí mezi nimi.

Contents

List of Figures

Chapter 1

Introduction

An option to undo user's actions is very important in graphical user interfaces since it is so easy to make mistake. Just a mouse click is enough to commit an action and frequently the user does not have an idea what she has just done. An undo feature implemented in a program enables studying, learning and exploration of the program without worries. This is especially important in the world of ongoing rapid changes and innovations of software.

Imagine a user who needs to create a paper with text, a table and a picture in a word editor such as Microsoft Word. After writing the first paragraph he makes text bold, inserts a table, types a second paragraph, adds a picture and inserts cells into the table. So far, everything has gone smoothly, but after some time, the user decides that the bold text is not exactly what he wants so he decides to undo it. The easiest way to accomplish this is to use the undo option in the program menu where the undo is implemented as a linear undo. After doing so, all work, except the first paragraph, has been undone with the formatting (Figure 1.1). Is this linearity what users want from undo?

Professors Chris Fernandes and Aaron Cass have conducted research [CFP06] in this area. They sought a more natural or user friendly approach to the undo process by conducting several studies and empirical evaluations. The results of the evaluations confirmed their initial hypotheses, that most people preferred so-called cascade undoing. The cascade undo allows users to take-back whatever step has been done during work without being concerned about losing steps that were done either before of after this step. If we return to the previous example, the problem with undoing the bold formatting would not be a problem anymore with this new approach. With the cascade undo, the user would be

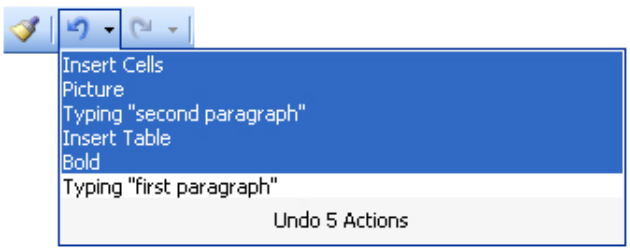


Figure 1.1: Undoing (linear undo) bold formatting and all following steps

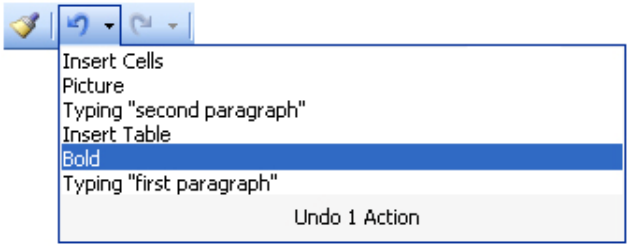


Figure 1.2: Undoing only "Bold" formatting (the cascade undo)

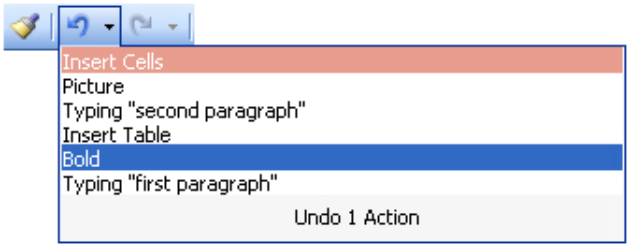


Figure 1.3: The cascade undo follows relations between the actions which are about to be undone. blue - the undone action, orange - the action undone because of the relation

able to undo only the bold formatting step, while maintaining the integrity of the rest of the work (Figure 1.2).

The cascade undo action derives its name from the fact that it does not merely undo one particular action but also automatically applies the same procedure to related actions. Let us return to the previous example and consider what would happen with the document if the user undid the table. The table would obviously disappear, but there is the "Insert Cells" action in history list. The cascading undo would just follow the relations between the "Insert Table" action and "Insert Cells" action in the history list and undo them both, as shown on Figure 1.3. Therefore we can rely on the cascading undo to simply transform a document from one stable state to another without affecting other work.

The linear undo is easy to visualize. There are two basic forms of its look in the current software. The first does not actually have a visual form at all (Notepad) and is just a menu item, while the second form is realized by a history list where users can see what action(s) will be undone. Unlike the linear undo the cascade undo is not as easy to visualize. We don't yet know how it should look or what features an interface should have in order to be the most natural for users.

In this thesis, we want to develop few visualizations for cascading undo and conduct several empirical evaluations, which will help us to understand the user's preferences.

Our motivation is that this new concept of undo migt be a great contribution to computer users. It migth encourage the users to explore programs and develop a better understanding of the software. It might also eliminate worries about losing work by going back and forth in the history of actions taken during creating documents or projects, and it might help users to work much more efficiently, thereby saving time, effort and funds. The cascade undo is a new way to radically improve programs' interface versatility.

Chapter 2

State of the Art

In this chapter we will present implementations of contemporarily used undo commands and lay down basic requirements for cascading undo visualizations. We will overview some core approaches to visualizations of hierarchical structures, mainly focus and context, and discuss which visualizations are best suited for our purposes.

2.1 Linear undo visualizations

Most of the contemporary software applications which support an undo command use a *linear undo model* of this command. In the linear model, the most recently executed action is the one that is undone. When an application implements a history list, user can require some action A_i to be undone and this causes that all the actions recorded to the history list after the A_i will be automatically undone by the system. The history list feature is an improvement because the user does not have to undo all action one after other starting from the most recent one and proceeding through the history list till the desired action is reached. The history list gives to the user an easier way to interact with the program's environment.

2.1.1 Linear undo without support of history list

In first case, Figure 2.1, a user is just presented with an arrow, a menu entry, or just a keyboard short cut. We can think of this visualization as "no visualization" since the user has no visual feedback about what action is going to be undone and how the action relates to other actions. Such implementation is limiting and allows very poor exploration and experimentation.

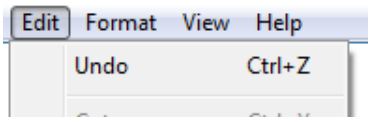


Figure 2.1: Linear undo in Notepad

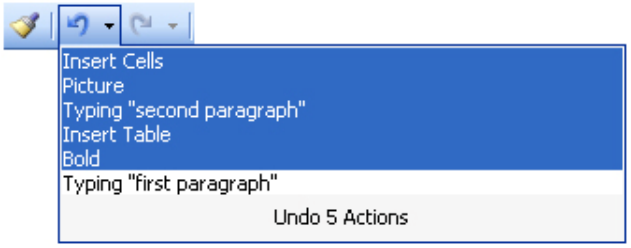


Figure 2.2: Linear nndo in MS Word with support of a history list

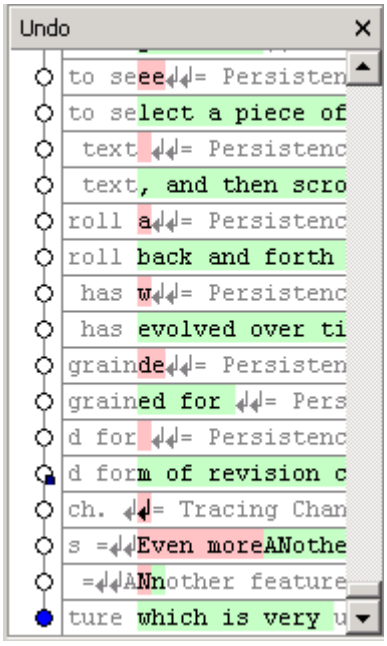


Figure 2.3: E-TextEditor undo list

2.1.2 Linear undo with support of a history list

Unlike the "no visualization" the history list, Figure 2.2, brings an option of choice and overview of actions. It is more apparent what action(s) will be undone and how they are related to the rest of the document (order of action in history list).

2.1.3 Branching undo model

This model attempt to keep track of all changes performed in a document by branching the history. A visual undo history, Figure 2.3, has been introduced by [htuu06]

The undo history is shown in a separate window which is updated as a user types. The vertical line interlaid by small circles, on the left side, shows the history of the changes. Each circle represents the document as it was at a specific point in time. By clicking on a circle the user can revert the document to the previous state.

On the right side of each circle is a one-line summary of the changes that leads to its current state. Changes are indicated by colors: green for insertion and red for deletion. It is put in context by the gray text surrounding it. The window is resizable so user can

easily enlarge it to see more of the change.

The solid blue circle indicates the user's current position in the history and the small marker (in dark blue) indicates when the document was last saved.

2.2 Non-linear undo visualizations

To reach better flexibility in undo the *non-linear undo model* has been introduced. This model allows a user to undo an arbitrary action A_i from a history list without influencing commands recorded to the history list after the action A_i.

Further we can divide a non-linear model to the *script undo model* and *selective undo model*. The only difference between these two models is the way how they treat dependences among actions of the history list.

2.2.1 Non-linear script undo model

In the script model [JEACS84] dependencies are ignored and an action, which is desired to be undone, is just taken out of the history list and all other actions are executed in the order they have been recorded to the history list. This can cause instability and unpredictability in the document.

2.2.2 Non-linear selective undo model

On the other hand the selective undo model [Ber94] takes the dependencies into the account. If a user tries to undo an action that involves some dependencies a warning is issued as a system response. Thus stability of the document is ensured.

2.3 Summary

This chapter overviews the current state of the "undo command world". The linear and non-linear visualizations are presented, followed by examples. In following chapter we will briefly explain what cascading undo is.

Chapter 3

Analysis of Visualization Techniques for Hierarchical Structures

In this chapter we will explain and overview some core principles for visualizing hierarchical structures which might span wide. These structures can generally contain and display big amount of information in a single view and still take up considerably small space.

But first, we will briefly explain the core concepts our work build upon, cascading undo and cascade.

3.1 Cascading undo

Cass, Fernandes present a concept of a *cascading undo model*[1] at [CF06]. Unlike other undo models, this model uses dependencies among the actions of the history list to cause dependent actions to be undone along with the selected action. The set of the actions, which are about to be undone together with the desired action, is called *cascade*.

3.2 Cascade

A cascade is formed by dependent tasks that form hierarchical structure with relations of parent - child, as on Figure 3.1. Since there cannot occure any backward dependencies, causing cycles, the structure is a tree.

If a parent is undone, all its child nodes follow it. As an example of the cascade, let us suppose that action A_{i+2} depends on an action A_{i+1}, and the action A_{i+1} depends on an action A_i. The cascade of the action A_i consists of the actions A_{i+2} and A_{i+1}, even thought the action A_i is not directly related with the action A_{i+2}. Thus, one can define the cascade as the transitive closure of an action.

[1] Also referred as cascading selective undo model.

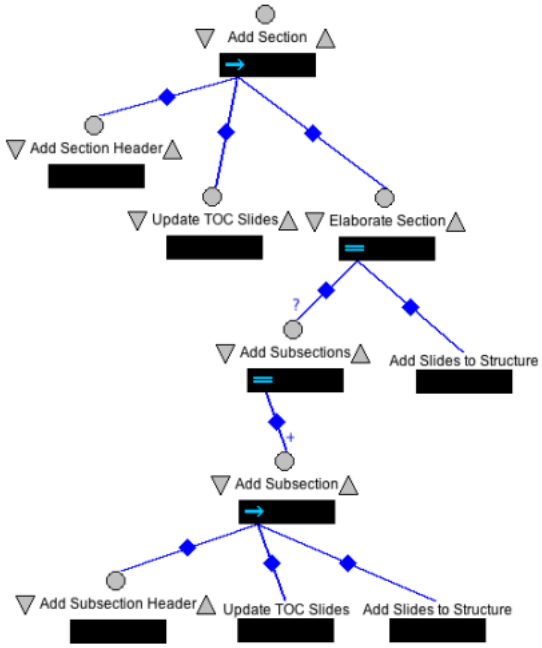


Figure 3.1: An elaboration of "Add Section" step. [CF06]

3.3 Requirements for visualizations

When thinking about cascading undo visualization few questions arise:

1. What are we actually going to visualize?
2. Where are we going to visualize it?

At the first place we have to realize that a cascade of an action A_i can grow arbitrarily long. Thus it can contain none, one, or hundreds of subsequent actions. The answer to the first question is that we are going to visualize a list of actions where the number of actions changes and the actions form the hierarchical structure [CF06].

Since the undo mechanism we are interested in happens on a computer the answer to the second question is a computer screen. However a computer screen where the cascade will be displayed is a limited resource. This implies that visualization has to be capable of presenting wide range of actions in limited space.

Thus, there are several requirements for the history list visualization to meet.

- intuitive to use
- takes little space on a screen
- provides low error rate during task selection
- performs fast
- ability to display big amount of information

Provided these limitations and requirements, we can now choose the right technique to perform the visualization.

All the visualization, we will examine in following text, are the focus and context visualization, or at least have some elements of this technique.

First, let us explain what the focus and context is and how it might be useful in seeking for the right visualization of cascading undo.

3.4 Focus and context

Focus and context is a technique of a visualization of information. Its principles are to show the most important data, the focal point (the focus), at great detail and full size, and oppositely to show the area around the focus, the context, to help to make sense of how the important information relates to the rest of the data structure.

To express relevance of data towards the focal point we can introduce *Degree of Interest (DOI)* and define it as a function of some relevant information of each piece of data to the rest of the data, and actually to any other factor. For instance the factor might be distance of data from a focal point, a priori importance, color, size .... DOI can be expressed as a relation among some relevant factors of data visualization, 3.1.

$$DOI(x) = F(x, x_i, x_{i+1}, x_{i+2}, ...) \tag{3.1}$$

The function F might take into account many factors and produce different results. Based on the result and original appearance we can divide focus and context techniques to subgroups such as *non-distorting* and *distorting*

The non-distorting subgroup does not distort the original perception of the data. Only part of the data is shown at the focal point and the rest of the data is not displayed. The awareness of context is provided by some kind of overview. An example of the non-distorting subgroup is a map. One sees just a little part of the map but at the same time is provided a map overview on the first page of the map book. Thus he knows where the detailed piece belongs to.

Predictably the distorting subgroup does distort the original perception of the data. At each moment of the visualization all data is presented but just a small group of it is in high detail, focus. In this case the context is given by position of the data in the view. In order to fit the focus view into the context view distortion is usually needed. As an example we can use a photography technique called fisheye. Here a photographer can capture wide angle views but only part of the view is in great detail (focus), the rest is distorted to sides of the photography (context).

Here are several focus and context techniques where each of them uses DOI in slightly different ways. We can divide these techniques into non-distorting and distorting subgroups. Here is a list of most significant ones:

Here we present several representatives of non-distorting and distorting views:

non-distorting views:

- separated views

- dynamic interactive zoom views
- lists

distorting views:

- perspective walls
- document lenses
- cone trees
- botanical trees
- hyperbolic trees
- fisheyes
- spirals

As we can see there are many visualizations leveraging focus and context technique. In following sections, we will overview some of them. We will discuss their suitability for the visualization of the cascading undo.

3.4.1 List

A list is the only one non-distorting focus and context visualization we will consider for cascading undo visualization. Lists are all over the web and at all kinds of software. Their main advantages are that they can be set to small size and still contain a lot of data, and they are really intuitive for a user.

Focus of a list is the entire area of the list. Context is provided by a size and position of a scrollbar. Thus a user is aware about a position in the list and its size. As we can observe on the figure 3.2

The list visualization is worth to further exploration since it fulfills the basic requirement for the cascading undo visualization.

3.4.2 Bonatree

Bonatree is a method for the visualization of huge hierarchical data structures presented at [EK]. The method is based on the observation that we can easily see the branches, leaves, and their arrangement in a botanical tree, despite of the large number of elements. Output of this method applied on a file system is on Figure 3.3.

Although this method is promising for the visualization of huge hierarchical data structures it is not suitable for usage in application such as word processing editors. It takes too much space and it is not really intuitive for a user. Also computation of this method takes a great deal of time since it uses 3D model.

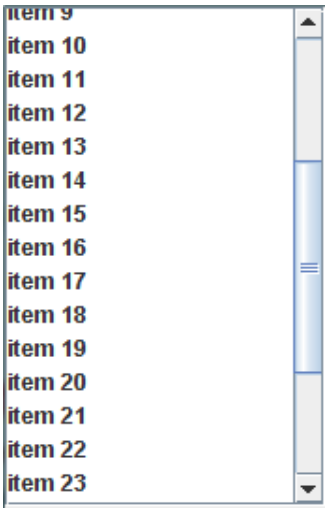


Figure 3.2: Java Swing JList with 30 items (12 focused)

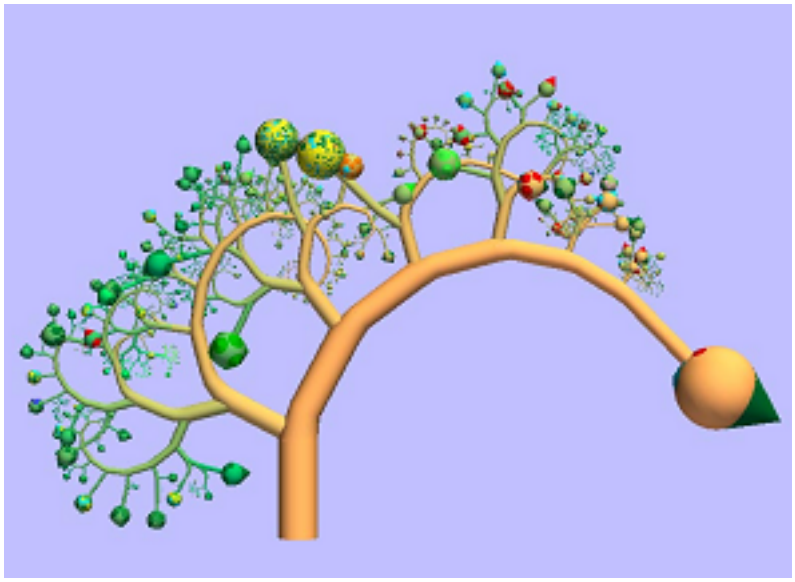


Figure 3.3: Hard disk file system visualization by Bonatree

3.4.3 Cone tree

The cone tree visualization presents a hierarchy in 3D to maximize effective use of available screen space and enable visualization of the whole structure. Interactive animation is used to shift some of the user's load to the human perceptual system [RMC91]. See Figure 3.4

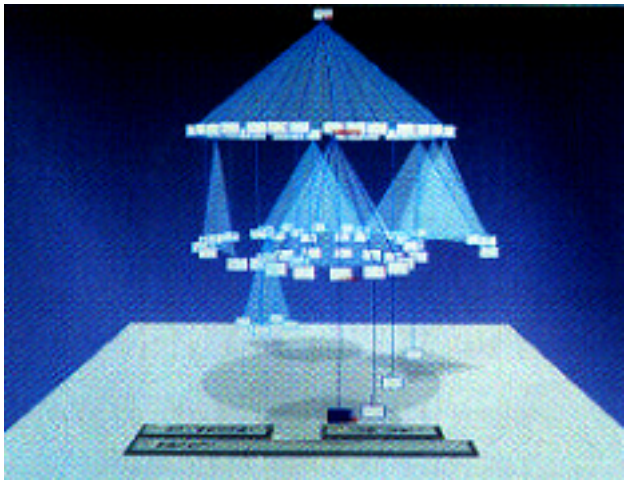


Figure 3.4: General hierarchy shown by the cone tree

This visualization has same obstacle as the bonatree visualization and thus is not appropriate for the cascading undo and its visualization in normal computer software.

3.4.4 Hyperbolic tree

Hyperbolic tree is another focus and context scheme for visualizing and manipulating large hierarchies. The aim of this approach is to lay out the hierarchy uniformly on the hyperbolic plane and map this plane onto a circular display region. The structure can change focus by translating the structure on the hyperbolic plane [LR94]. See Figure 3.5.

Unlike bona tree and cone tree the hyperbolic tree is a 2D technique. It would be easier to integrate it to a software environment. However it is space demanding as well which disqualifies it as a candidate for a cascading undo visualization.

3.4.5 Fisheye

Fisheye is another interactive focus and context technique that shows local details and global context in the same view. Fisheye magnifies and compress parts of the view depending on actual view point. In other words we are presenting only the most interesting sub-view from the entire view. DOI takes into account both the *A Priori Importance (API)* of items in the view, and their *Distance (D)* from user's current focus. We can form a following function for fisheye degree of interest at some point x, given the current focal point x_0,

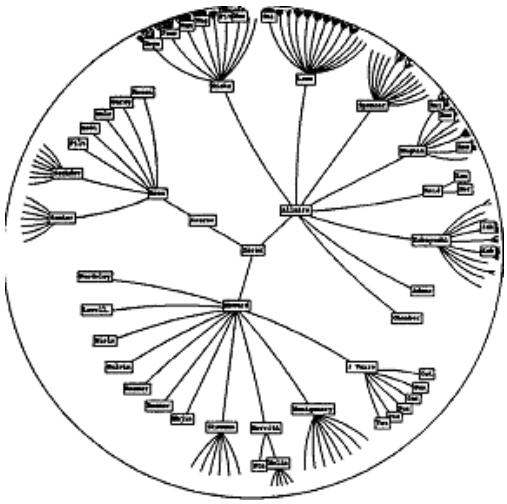


Figure 3.5: An organization chart displayed by a hyperbolic tree

$$DOI_{FE}(x, x_0) = F(API(x), D(x_0, x)) \tag{3.2}$$

where F is some monotone increasing function in the first argument, and decreasing in the second. So the DOI reflexts an item's importance and position to the current focus as shown on the figure 3.6.

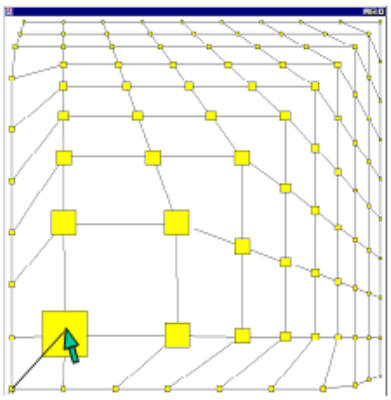


Figure 3.6: A grid that is distorted according to the DOI function.

We can define a threshold t value such that item's DOI must be great that t in order to be displayed. This help to exclude non-important items from the view to increase transparency. Refer to [Gut02] for more details on this topics.

The fisheye, thanks to its great spatial and focus and context features, is a great candidate for visualizing the cascading undo and its cascade.

3.4.6 Spiral

Generally speaking size of a screen of a mobile device is really limited. On the other hand number of contacts in a contact list of such device can grow very long. Huot and

Lecolinet [HL06] describe a usage of a combination of a spiral and a focus and context technique for a mobile device contact book, as shown on the figure 3.7.

Figure 3.7: A SpiraList displaying 100 contacts on a VGA PDA

One can easily imagine a cascading undo history list mounted to this visualization. It perfectly fits the spatial requirements and so becomes a good option for further reflections.

3.4.7 Others visualization

Naturally we have not mentioned all the possible visualizations. The reason for this was that they were not as promising as those we mentioned above. Here is a list of few other visualization approaches about which we initially were thinking that would help us with bringing the cascading undo to a user. For completeness we list them here:

1. document lenses, [RM93]
2. perspective walls, [MRC91]
3. dynamic interactive zoom views, [M.J]

3.5 Summary

This chapter explained what *focus and context* principles are and how they relate with the requirements for cascading undo visualization. Some core focus and context visualization has been overviewed and briefly described.

The next chapter will built up visualizations of the cascading undo based on the visualizations above and analyze their positives and negatives.

Chapter 4

Design of Cascading Undo Visualizations

After a thorough comparison of the visualizations and their features in the previous chapter, this chapter proposes several models adapted to cascading undo. We will present these models and their usage in great detail and thus lay down the ground for the future implementations and evaluations.

There are common requirements for all visualizations. Beside the requirements from previous chapters they all have to be capable of

- presenting and browsing a history list of actions,
- letting a user to choose an action to undo,
- showing and reviewing a cascade of an action to be undone and
- committing the undo command of the actual action and its cascade.

We have opted visualizations which do not show a hierarchical arrangement, except the highway visualization, of a cascade[1], since we believe that it would be confusing for users to see how various cascade items relates. It would be hard, as well, to fit such a hierarchical representation within a limited space of programs' environments. It is important to bear in mind, that an undo is just an integrated feature of a program and thus it should be an "invisible" part of it. This is why following models show a cascade as a set of actions not as a hierarchical structure - to keep everything as simple as possible.

4.1 Simple list

The simple list model is base on the visualization technique of *list* discussed on page 10 at section 3.4.1.

A list is not capable of showing an entire history list at single view and a scrollbar must be used to allow scrolling through the history list. The scrollbar also provide a visual reference of position in the history list and its length (non-distorting focus and context).

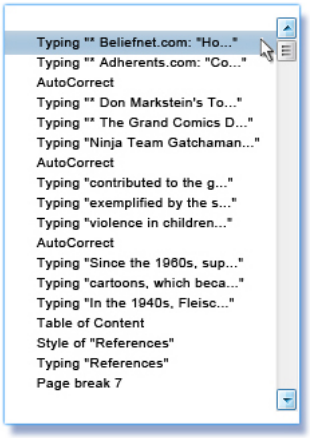
[1] See [CF06] for the explanation of the hierarchical aspect of the cascade.

The simple list model works at two modes, *history list browsing mode* and *cascade browsing mode*. The former is normal list browsing and it has already been discussed, the latter is activated by clicking on an item whose cascade we are interested in.

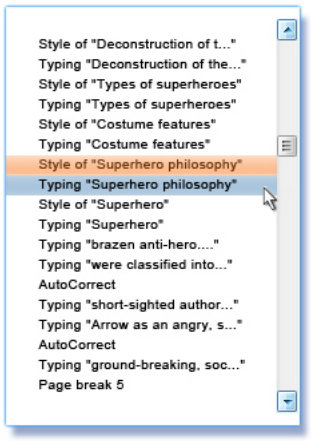
When the list is in the cascade browsing mode the entire cascade of the clicked item is highlighted and stays highlighted till another click is made. Thus a user can review the entire cascade by moving the scrollbar without worries that the highlighting would disappear. Double clicking on the selected action undoes it and the item cascade.

4.1.1 Usage scenario

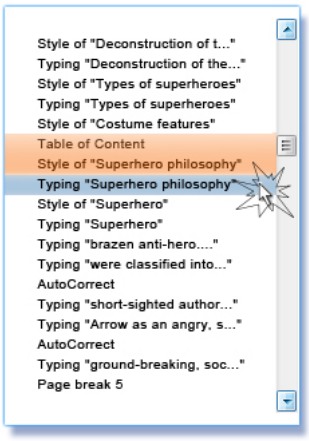
1. History list browsing mode
An item is highlighted.



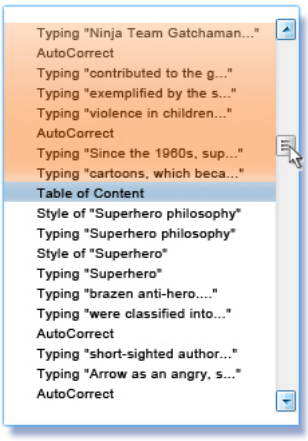
2. Selecting
By scrolling, the history list is browsed. The hovering mouse highlights the cascade.



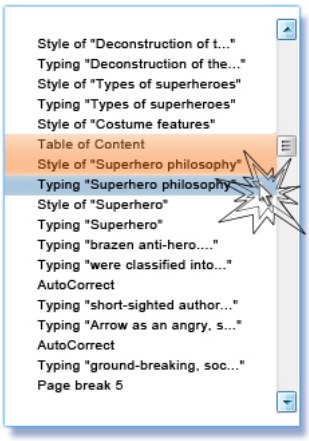
3a. Cascade browsing mode
By clicking the blue highlighted area the cascade appears and can be browsed.



3b. Cascade browsing mode
The part of the cascade is shown and the rest might be viewed by scrolling. If clicked again view switch back to normal.



4. Committing undo action
An item and its cascade is undone by a mouse click on the item. .



We believe that the simple list visualization will be well accepted by users. Mainly due to the fact that the list concept is familiar to most of users thank to its widespread usage. The only problem might be the actual understanding of the cascade browsing mode, as its control is not trivial.

4.2 Fisheye

A fisheye visualization is inspired by "fisheye menus" described in [Bed00]. This approach utilizes traditional fisheye graphical visualization technique to linear menus, as decrypted at section 3.4.5 on page 12. An effective mechanism is provided to select items from long lists which appear in the cascading undoes. Fisheye dynamically changes the size of list

items to provide focus area around the mouse pointer. In such way it is possible to display the entire list on a part of a single screen without additional controls for navigation.

When the list of the actions is explored by a mouse cursor a cascade of the active item is highlighted. However, when a user desires to explore the cascade by moving the mouse toward it, the focus changes and the former cascade is replaced by new one, belonging to different item. To allow users to see what actions are about to be undone another mechanism besides the highlighting is needed. We present a view of the cascade in a separated detail window placed right next to the item in focus. As the cascade's size is unknown beforehand, the size of the detailed windows might be in extreme case same like the size of the entire original list. This is a reason why the detail window uses fisheye mechanism for presenting the items of the cascade. Thus we ensure that the cascade will never be bigger that its parent control.

The cascade might not be continuous at its entire length. Since ts items might be interlaid by items of a completely different cascade. We attempt to address this issue by various methods. The ellipses or the miniatures of the interlaid action might be inserted between the separated parts of the cascade, as shown on Figure 4.2.1.

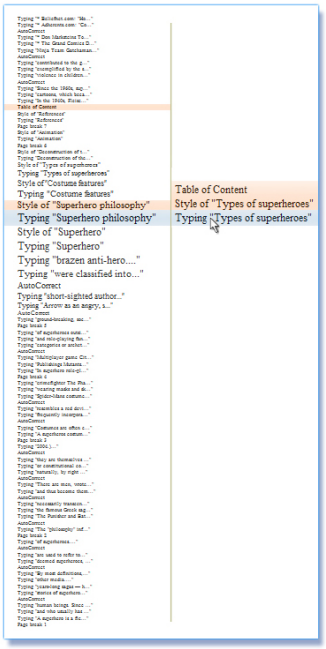
4.2.1 Usage scenario

1. Browsing the history list

The focus area changes by moving the mouse. When the mouse reaches an intended point in the history list the cascade highlights and the detailed window pops up on the right side

2a. Viewing the cascade - no separation

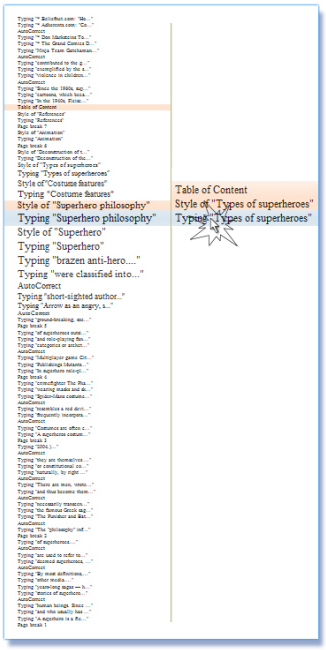
The items in detail window are treated at fisheye manners. The individual parts of the cascade are connected.



2b. Browsing the cascade - ellipses
The items in the detail window are treated at fisheye manners. The individual parts of the cascade are separeted by ellipses and coloring.

2c. Viewing the cascade - miniatures
The items in detail window are treated at fisheye manners. The individual parts of the cascade are separeted by miniatures of other actions.

4. Confirmation
The click on selected item the undo.



The fisheye visualization employed within the cascading undo seems to be a promising alternative for further implementations. We expect it to be well accepted by users.

Although, it might seem to be slightly strange to some users at the beginning.

4.3 Spiral

The spiral visualization was inspired by [HL06], and is shown on Figure 3.7 on page 14. The idea of winding of a history list onto a spiral provides space non-demanding means for presenting long lists. However, in the original usage of the spiral (a contact list for mobile devices) the list items were grouped by first letters in non-focus area, closer an item was to the focus area more detail of it could be spotted. Unfortunately we cannot perform grouping with cascading undo items since the criteria for grouping is vague. Also there is a problem with browsing through the list. Since spirals are not enclosed structure as circles the items would eventually fall out of it in one of the ends, depending on the direction of moving. It is no clear what should happen with these items.

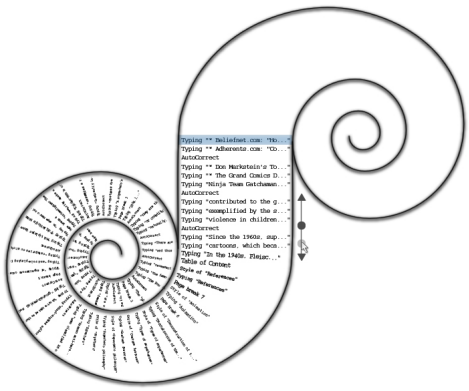
So we propose a system of two connected spirals working together. One spirals "sucks" items coming from one direction and vice versa. Initial state leaves one of the spirals empty whereas the other contains all the item of the history list winded up on it. By browsing the items move and rewind to the other spiral. This reminds a video cassette for VHS. Still the spirals have limited space for its items, the reason is fixed number of convolutions. Eventually, some items would have to be omitted to display in the either end of the spiral system.

A cascade of an item is shown by hovering of the mouse cursor over the item. After a click of the mouse the cascade's highlighting is locked and the cascade's items might be reviewed.

4.4 Usage scenario

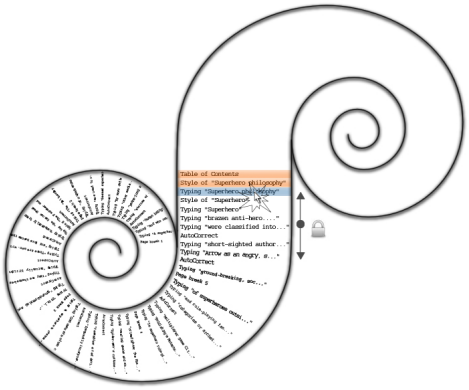
1. Browsing
The first item is highlighted. By dragging the scrollbar up and down the focus area changes. The speed of rolling depends on the deviation of the button from its initial middle position.

2. Selecting
When an intended point in the history list is reached the scrollbar is released and the mouse is moved to the spiral where mouse hovering over an item picks the element and highlight the cascade.



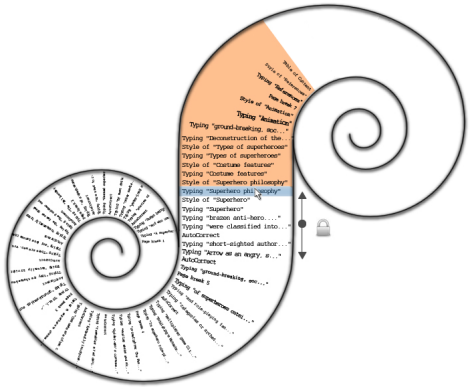
3a. Viewing The Cascade

By clicking the blue area only the cascade is highlighted (the items not in the cascade disappear). The view is locked on the cascade and thus the cascade can be browsed by the scrollbar. A little lock sign appear and thus indicates the change of the state. To unlock the view (go back to normal state) the lock sign or the highlighted cascade must be clicked once.



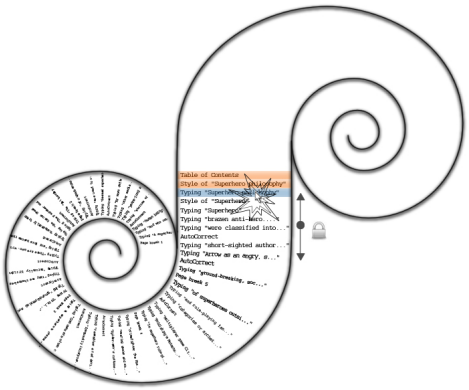
3b. Viewing The Cascade - extended

The cascade can be browsed as the history list in first step with the scrollbar. The blue area stay highlighted for entire browsing and so provides visual clue where the cascade begins.



4. Confirmation

The double click on the blue item or the highlighted cascade confirm the undo command.



We have serious concerns about its complexity both, for programmers and for users, especially because of legibility of the items winded on the spirals. We believe that this method is not as promising for our purposes as we initially hoped.

4.5 Highway

This model utilizes the hierarchical property of a cascade and a proposal of it is on the figure 4.1. It aims to show how items in the focus area are related. In the highway visualization the fisheye principles are used. Items out of the focus area dynamically change the size base on the distance from the focal point. The focus area consist of n columns, where n is number of items in the focus. To indicate that items are members of the same cascade they are placed to one column. To emphasize the order the items were added to the history list at they are mutually shifted in direction of adding.

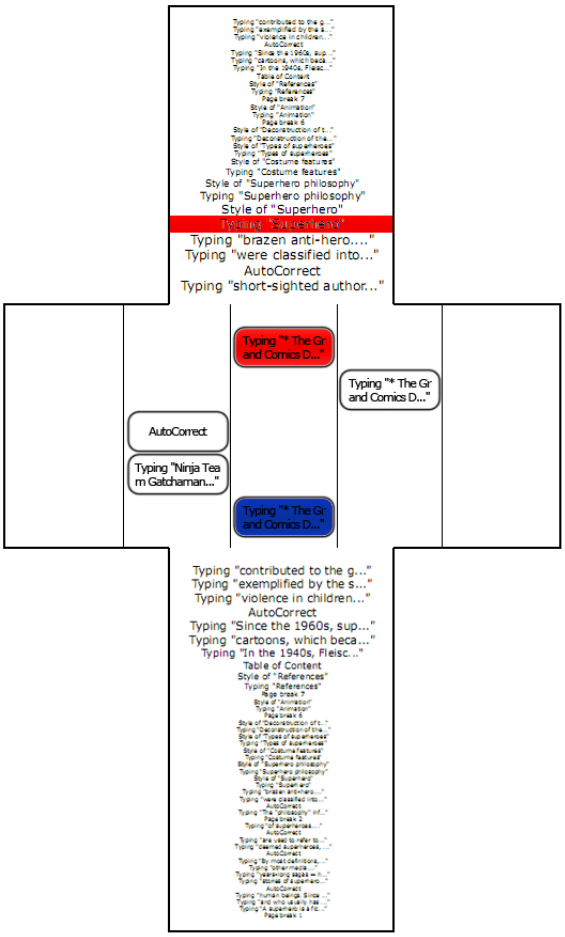


Figure 4.1: Highway undo visualization: An item and its cascade highlighted

The figure 4.2 represent the highway focus area (size $n = 5$) with five column. In the first (left) case all five items are independent and they do not belong to same cascade. Whereas in the second (right) case item no.4 depends on item no.2, and item no.5 depends on item no.3. Two lanes remain empty. The focus area reminds a highway lanes with items scattered over the lanes (origin of the name highway).

The highway model is the only model which takes into account the hierarchical structure of the cascade. We assume that it will be hard to be understood by users. Also, the space taken by this visualization is not small, especially if higher number of cascades is displayed at the focus area. There is no state diagram for highway model since this implementation will be implemented just to make sure that our observing about its

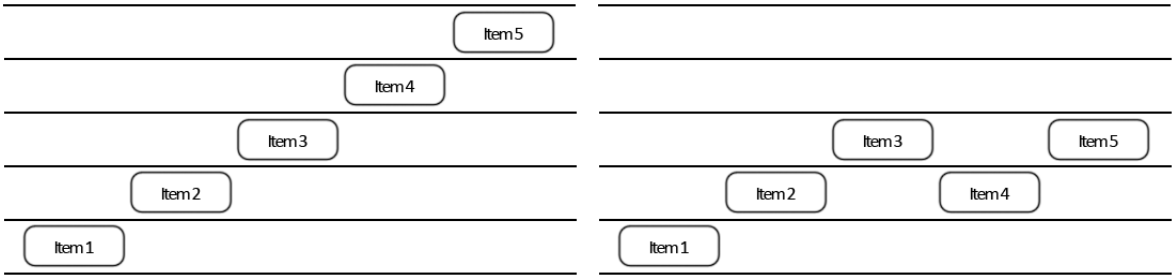


Figure 4.2: 5 items in the highway-like arrangement, left: 5 independent cascades, right: 3 independent cascades

usefulness is right

4.6 Summary

This chapter proposes several scenarios for visualization of the cascading undo, namely Simple list, Fisheye with detail, Fisheye with scrollbar, and Highway. These scenarios aim to be intuitive and easy to use.

The next chapter overviews implementations of some of these scenarios.

Chapter 5

Implementation

In order to examine which visualization is the best for users an evaluation is needed. To conduct the evaluation we either could construct some models or program implementations of the visualization proposed in the chapter 4. Due to complexity of the proposed visualization, constructing the models would be too difficult, if not impossible. Thus we decided to program the implementations. In this chapter we will overview these implementations. Finally we have programmed following undoes:

- Simple list
- Fisheye with detail
- Fisheye with scrollbar
- Highway

They slightly vary from the proposed visualization but the main aspect remains. All implementations are written in Java 1.6 and they inherit from JComponent. This allows placing them in either JFrame, or another JComponent.

In general the implementations are built of blocks: History List, Visualization Manager, Measure, as you can see on Figure 5.1. In the remainder of this chapter we will overview the individual blocks and then the implementations them self.

5.1 History List

To manage dependencies among the items we use an instance of *HistoryList* class in all implementations. The instance of this class keeps track of history list items and their dependencies in recursive manners. Its methods work with cascades of the items and serve to classes handling visualization. This class is identical in all implementations.

An instance of the History List composes of History Items. All the History Items are stored in *ArrayList<HistoryList>*. An History Item might be added by method *add* or in the constructor of a History Item by specifing the History List.

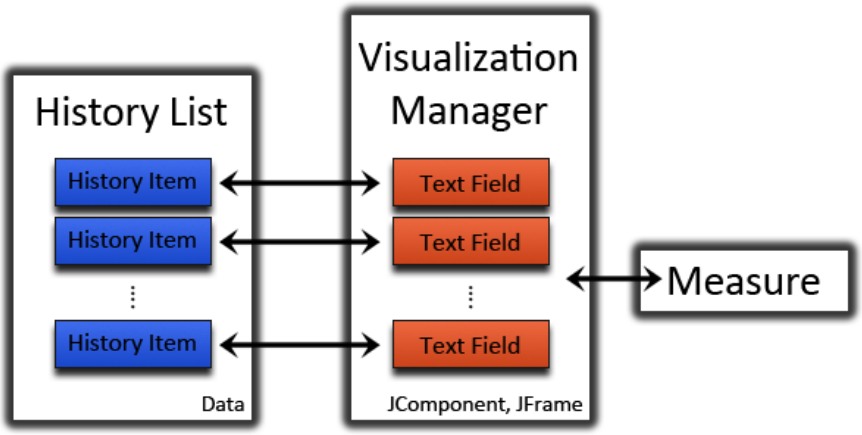


Figure 5.1: Programmatic structure of the implementations

5.2 History Item

History Items holds its text, which is displayed, and the cascade, if there is any. The cascade field is *ArrayList<HistoryList>* and only contains direct children in the cascade hierarchy. The direct children are added to the cascade field by method *addToMyCascade*. When the full cascade is needed, the method *getCascade* should be called, and transitive closure is computed. Thus we obtain all the items which are directly and even indirectly dependent on the particular history item.

5.3 Text Field

The Text Fields are the basic elements of the visualizations. They dispatch mouse events, render the text, perform the highlighting and change their dimensions as needed. There is has-a relation between Text Field and History Item. Thus every Text Field may leverage the method of the History Item and History List.

The Text Field extends JComponent and implements MouseListener. When it is being inicialized an instance of History Item and the width (in pixels) of the field are passed as the input parameters. The actual dimensions and the text position is set by calling method *refresh* from the Visualization Manager with following input parameters: the field height, the text position, and the font.

When the mouse cursor enters the instance of the Text Field, it is set to focus by calling method *setActive*. Then the method of the History Item *getCascade* is called and the Text Fields which correspond with the History Items in the cascade are highlighted by calling their method *setHighLighted*. The oposite is achived by calling method *setPasive*

5.4 Measure

To be able to draw the History List, the exact size and position of the text within a Text Item has to be determined. Class Measure takes as input parameters the maximal

windows height, the number of items in the history list, the number of items in the half of the focus area (focus zone radius), the gradient function describing the font-size difference to the focus, and the font sized for the focus area.

The Measure computes the size of items in the context, respecting the focus and the gradient sizes. The context items size is computed so the implementation takes up as much space as possible.

In order to get necessary information for rendering an Text Item, there are provided methods *getFontSize*, *getTextPos*, *getFieldHeights* taking as the only one input parameter the item's distance from the focus and returning respectively font size, text position, and Text Item Height.

We have to add that the previous happens only in the fisheye based visualizations, the Fisheye with Detail, the Fisheye with Scrollbar, and the Highway. In the Simple List implementation, there are fix values of font size, text position, and Text Item Height. Thus, in this case we can implement the measure just by 3 variables.

5.5 Visualization Manager

As the name suggests the Visualization Manager manages the entire visualization. It creates JFrame, initialize and layouts the Text Boxes, initializes the Measure, the History List and so forth. To avoid mislead we have say that this is not a single class, but three classes. However there is always one crucial class in every implementation. This class is called in

- Simple list - Scroller
- Fisheye with detail - Fisheye
- Fisheye with scrollbar - Fisheye
- Highway - Highway

All of these classes have method *refresh*, that takes one integer parameter, the index of the active item. This method is called when focus changes. As it know the current focus point, it can use the data computed by the Measure object to render the entire cascade by calling *refresh* method of every Text Field.

The two other classes only initializean object of previous classes and place it to the JFrame with the simulation of the real program for the evaluation. They are inherit

5.6 Simple list

Simple list is based on the visualization with the same name, Simple list (page 15, section 4.1). They are no major difference between the proposal and actual implementation.

The implementation is straight forward and leverages JSrollPane. The *refresh* method just iterates over all Text Fields and set them passive or active.

The final look and the various stages of the Simple List are shown on Figure 5.2.

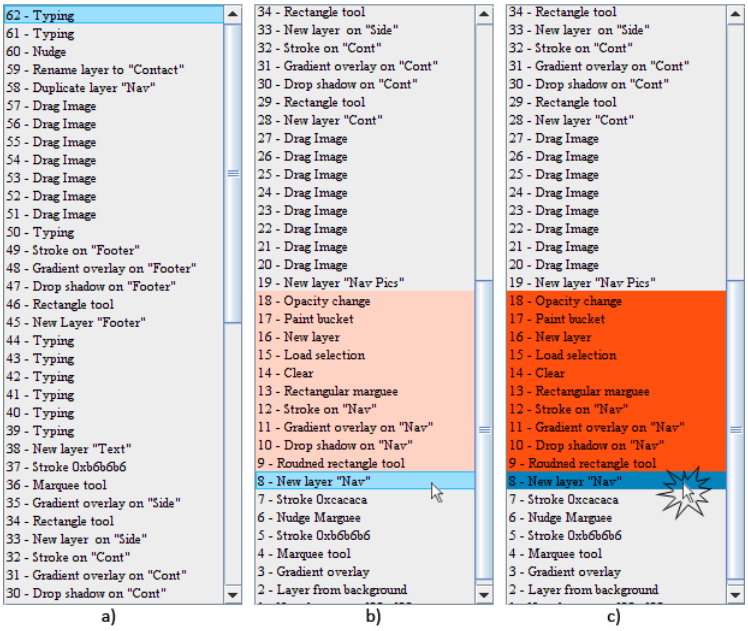


Figure 5.2: Simple list: a) browsing the the list, b) highlighting the cascade, c) switching to the cascade browsing mode

5.7 Fisheyes

Fisheye with detail, Fisheye with scrollbar and Highway are all based on the fisheye principles. They differ in the way how they visualize the cascade at the focus zone.

5.7.1 General principles

In a fisheye visualization, all of the items of a history list are in a single view that is completely visible. The items near the mouse cursor are rendered at the full size, but items further away are rendered at a smaller size. Items of the history list are dynamically scaled in such way that the focus area around the mouse cursor form a “bump”.

There are four general parameters that a programmer can control: focus area length, gradient between peripheral items and in-focus items, font size and maximum space taken by the visualization. The focus area length specifies the number of items rendered in full font-size as shown on figure 5.3. The gradient between focus and context is an integer array that defines the font size difference to the full font size for items in the gradient area. The maximum size is never exceeded by the fisheye and thus the size of a window is predictable.

Every possible *degree of interest (DOI)* is computed for each item in the history list, as described on page 12 in section 3.4.5, in advanced when the a fisheye is initialized. This ensures smooth behavior since no computation is needed. DOI counts only with distance of the item from the focus and not with any a priori importance. Fisheyes DOI function is drawn on Figure 5.4. The size of items in the focus and gradient area is specified by an application programmer and the size of the items in the context area is computed by the Measure object.

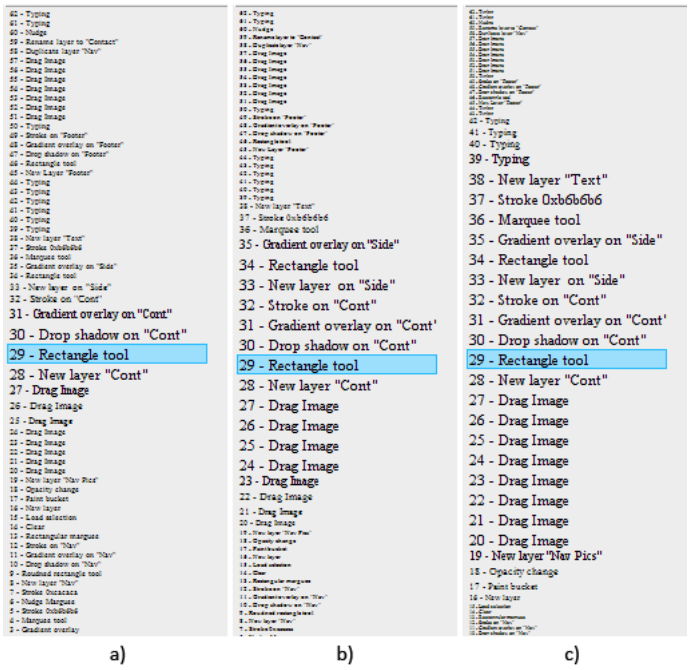


Figure 5.3: 3 ,11, 19 item in focus

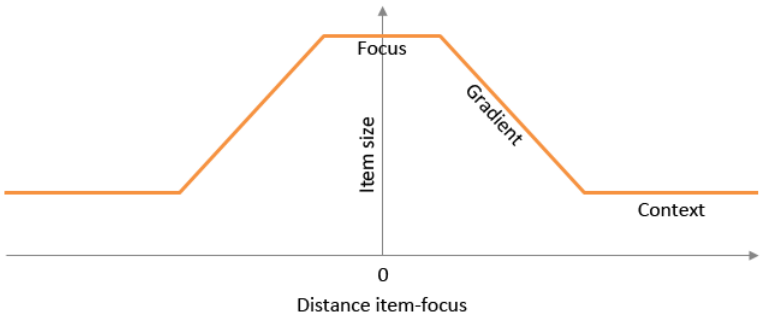


Figure 5.4: The Degree of Interest function determines the size of items. The fuction is implemented by the Measure object.

5.7.2 Stability

If the DOI is implemented according the Figure 5.4 everything works fine until the cursor gets near the ends of the fisheye, the "'without correction' part of Figure 5.5. Then the visualization becomes unstable and it makes the lower end [1] to move up and down. The cause of this is that we render each item based on the position of the item before it. If one item changes its size all the items, bellow it, are shifted up or down. Moving the focus in the middle of the fisheye does not cause any complications because for every item that gets bigger, another item gets smaller by the same amount.

To address this complication we need to realize that if the focus is on the first item, there is only half number of items rendered in full size and gradient area size, but there are more items rendered in the minimal font size than in the case when the focus is in the middle of the fisheye. As we move the focus to away from the first item the number of items in full size or gradient size grows and number of items in minimal font size gets smaller.

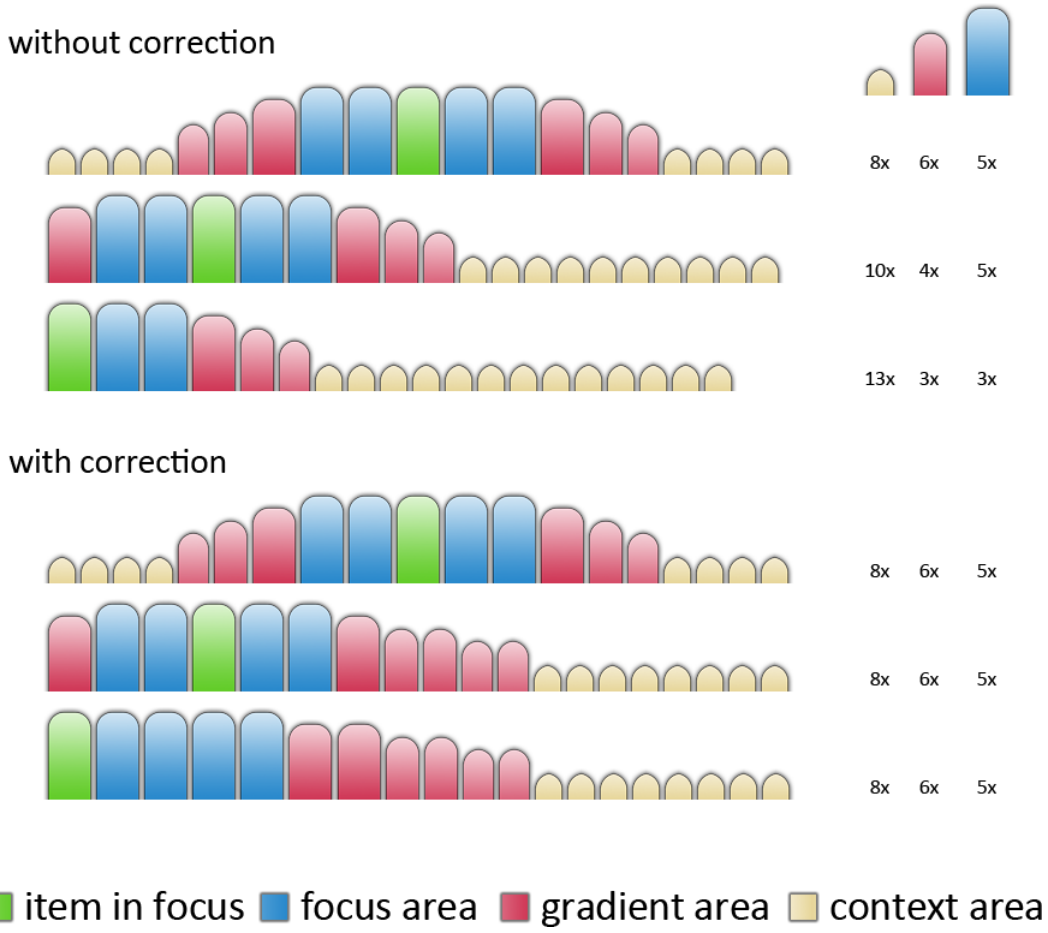


Figure 5.5: Stability of fisheye - The fisheye without correction changes its size base on the item in the focus, number of items is not distributed constantly. The fisheye with correction always distributes the items constantly and so ensure the stability.

Simply put, the number of item in various font size area changes depending in which

[1] On Figure 5.5 the lower end is drawn on the right side

part of the fisheye the focus is, see the number of items in the groups on Figure 5.5. To avoid the instability we fix the number of the items in each font-size area, so there is constant proportion of number of items with different font size, "with correction" part of Figure5.5. There the first image presents the layout of the items' font size when the focus is in the centre of the fisheye. The items' font size is evenly laid around the focus. When the focus moves to left, the second image, there is not enough space for the gradient area to fit in. If no correction is used the gradient area gets smaller, however the context are grows. With correction, the items, which would be rendered smaller, are rendered bigger, exactly in size of items which did not fit in. The same happens with the focus area, when pushed too much to the left end, as shown on the third image.

Here we show the actual code of the method *render* for the fisheye based implementations.

```
public void refresh(int itemIndex) {
        this.activeItem = itemIndex;

        if (measure == null) {
            return;
        }

        //set all items to min size
        int tmpEnd = measure.getArraysLength() - 1;
        for (int i = 0; i < fields.length; i++) {
            int tmpFieldHgh = measure.getFieldHeights(tmpEnd);
            int tmpFontPos = measure.getTextPos(tmpEnd);
            Font tmpFont = measure.getFont(tmpEnd);

            fields[i].refresh(tmpFieldHgh, tmpFontPos, tmpFont)
                ;
            fields[i].setPasive();
        }

        //set active item to active size
        int tmpActFieldHgh = measure.getFieldHeights(0);
        int tmpActFontPos = measure.getTextPos(0);
        Font tmpActFont = measure.getFont(0);
        fields[itemIndex].refresh(tmpActFieldHgh, tmpActFontPos
            , tmpActFont);

        //Set focus and gradient area. This area is symmetric.
            The for look goes from the center (active item) and
            place items evenly on both side, if possible.
        //If not it is not possible it places them to only one
            side and so deal with stability issue

        int up = 1; //how many items has been placed above the
            active item (itemIndex)
```

```
        int down = 1; //how many items has been placed bellow
           the active item (itemIndex)
        for (int i = 0; i < measure.getArch().length; i++) {
            int tmpPos = measure.getArch()[i];

            int tmpFieldHgh = measure.getFieldHeights(tmpPos);
            int tmpFontPos = measure.getTextPos(tmpPos);
            Font tmpFont = measure.getFont(tmpPos);

        //Can we place item bellow the actual item? Yes ->
           place it there; No-> place it above
            if (itemIndex + i + 1 < fields.length) {
                fields[itemIndex + down++].refresh(tmpFieldHgh,
                    tmpFontPos, tmpFont);
            } else {
                fields[itemIndex - up++].refresh(tmpFieldHgh,
                   tmpFontPos, tmpFont);
            }

        //Can we place item above the actual item? Yes -> place
            it there; No-> place it bellow
            if (itemIndex - i - 1 >= 0) {
                fields[itemIndex - up++].refresh(tmpFieldHgh,
                   tmpFontPos, tmpFont);
            } else {
                fields[itemIndex + down++].refresh(tmpFieldHgh,
                    tmpFontPos, tmpFont);
            }
        }
        fields[itemIndex].setActive();
    }
```

5.8 Fisheye with detail

This implementation is base on proposal of Fisheye visualization, on page 17 in section 4.2. It utilizes fisheye principles as described in previous section.

A cascade is shown in two ways, as a highlighted overview on the fisheye itself and in a separate window as a detailed overview. Since the cascade's size can grow big, we use a fisheye in the detail window as well. Thus there are two fisheyes utilized in this implementation. The one in the detail view is dynamically created when the mouse cursor enters an item and it is aligned with the item, it disappears when the cursor leaves the mouse.

The classes of the fisheye are *Lafe* and *Rafe*. Lafe stands for Left aligned fisheye, Rafe stands for Right aligned fisheye. Both of them inherit from the *FishEye* class.

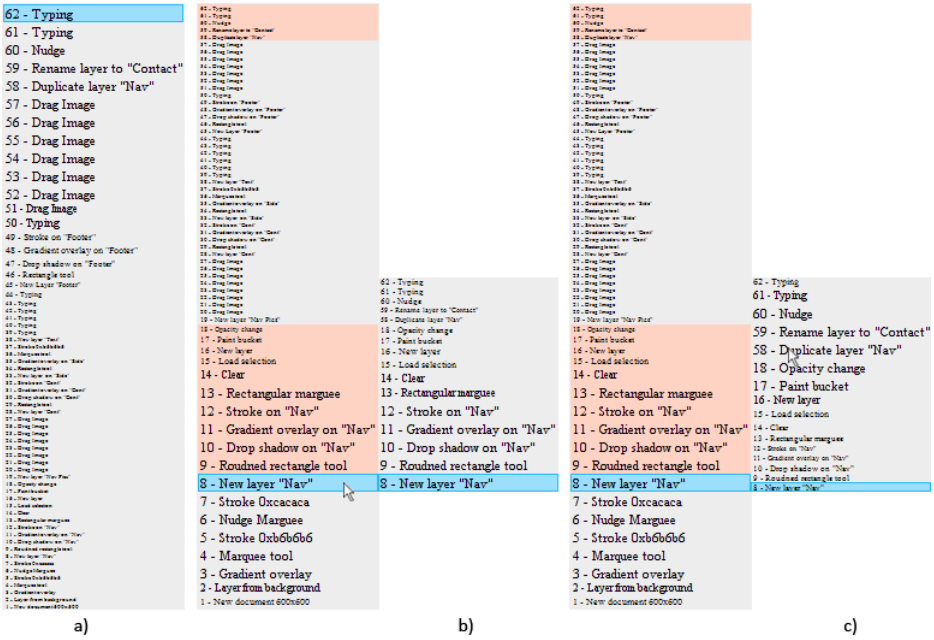


Figure 5.6: Fisheye with detail: a) browsing the list, b) highlighting the cascade, c) browsing the cascade

The *Rafe* fisheye is added and removed from the JComponent by method *addRafe*, *removeRafe* of the *FishEyeUndo* class, which belongs to the Visualization Manager category, and are invoked from the Text Field elements when the mouse cursor enters and exits them.

The final look and the various stages of the Simple List are shown on Figure 5.6.

5.9 Fisheye with scrollbar

This implementation is not base on any proposed visualization. It is a mix of Fisheye and List visualizations. It gives up the detail window but adds a scrollbar. The user's logic of works in similar way as Simple list visualizations. It employs two modes, *the history list browsing mode* and *the cascade browsing mode* as well. Switching between these modes is done by a click of the mouse as in the Simple list undo.

The scrollbar mimic the functionality of movements of the mouse cursor, it changes the focus.

This visualization is based on the fisheye principles as well. Its inner structure copies the structure of the Fisheye with detail.

We decided to implement this visualization since we believe it will be well accepted. It aims to be very simple to use and still capable to fulfill all requirements on the cascading undo visualization.

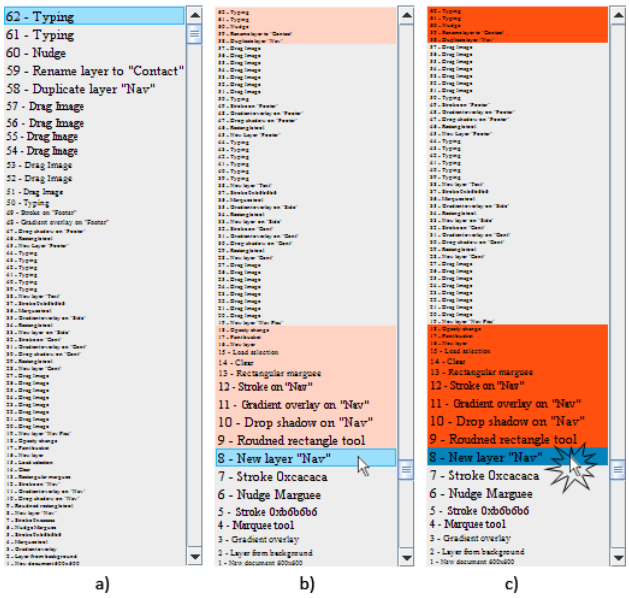


Figure 5.7: Fisheye with scrollbar: a) browsing the list, b) highlighting the cascade, c) browsing the cascade in the cascade browsing mode

5.10 Highway

Initially we did not want to implement the highway visualization. Finally we decided to give it a chance and implement it. The implementation does not aim to take part in future evaluations, however the important argument is that it is the only visualization that takes into account the hierarchical structure of the cascade. Thus, we will have more options for choosing the right variant.

The context and gradient area are identical with fisheye with scrollbar implementation. However the focus area differs and utilizes the highway principle, proposed at section 4.5 on page 22.

The *HwTextField* classes can either be display in normal manner of can be switched to the highway mode, where it plots the cascade hierarchy of the items in proximity. The Highway is managed by *HwBox* class.

In the first implementation we wrote, the items in the focus area did not stay in the lane where they entered the highway. Instead they flicked to another row with a change of the focus. This made it difficult to follow the items and their dependencies. The cause of this was the redistributing algorithm which alternated columns, where the items were put, based on the distance of the item from the focus item.

We solved this by assigning every item a default placement index, DPI. The DPI is computed by function 5.1:

$$DPI(x) = HLI(x) \bmod (HWL - 1) + 1, \tag{5.1}$$

where x stands for the history list item, *HLI* for history list index of an item and *HWL* for highway length - the number of columns in the highway area. Thus, if an item is not in the focus or in the cascade of other highway-area item, it always takes DPI-*th* column

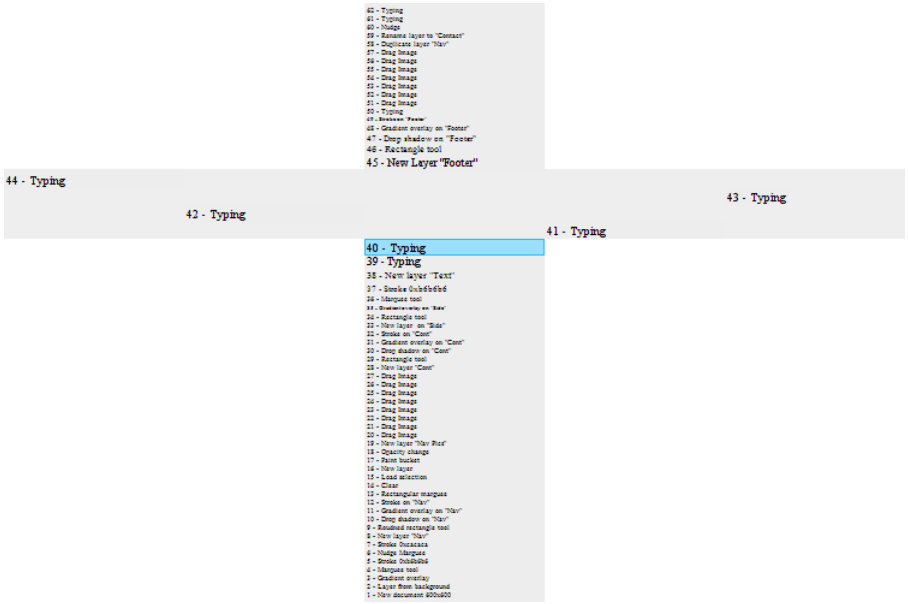


Figure 5.8: Highway spanning wide

regardless of the actual focus item.

We leave both methods responsible for the distribution in the code. The old one is *HwBox.renderBox1* and the new one is *HwBox.renderBox2*. The renderBox method is called from constructor of the class HwBox.

The highway takes too much screen space, as show on Figure 5.8, and is an inappropriate visualization for the cascading undo. Also the small highway area does not convey any useful information about the cascade and its member.

5.11 The evaluation environment

In order to conduct the evaluation we placed the Simple list, the Fisheye with detail, the Fisheye with scrollbar into the faked environment of the market leading graphic editor Adobe Photoshop CS3, Figure 5.9. Figure 5.9 and Figure 5.10 compares the evaluation and real environment

Committing a chosen action in the history list of the evaluation environment causes that the project changes according the undo philosophy. Thus, we simulate the effect of the undo.

The environment is built up from small images which are associated with items of the history list and stacked on top of each other. When an action from the history list is undone the images in question are removed. This makes an illusion of the real cascading undoing.

The method for rendering the evaluation *drawAllCascadeImages* is part of the History List class. This method loops over the History Items, retrieves the images associated with them, and draw them.

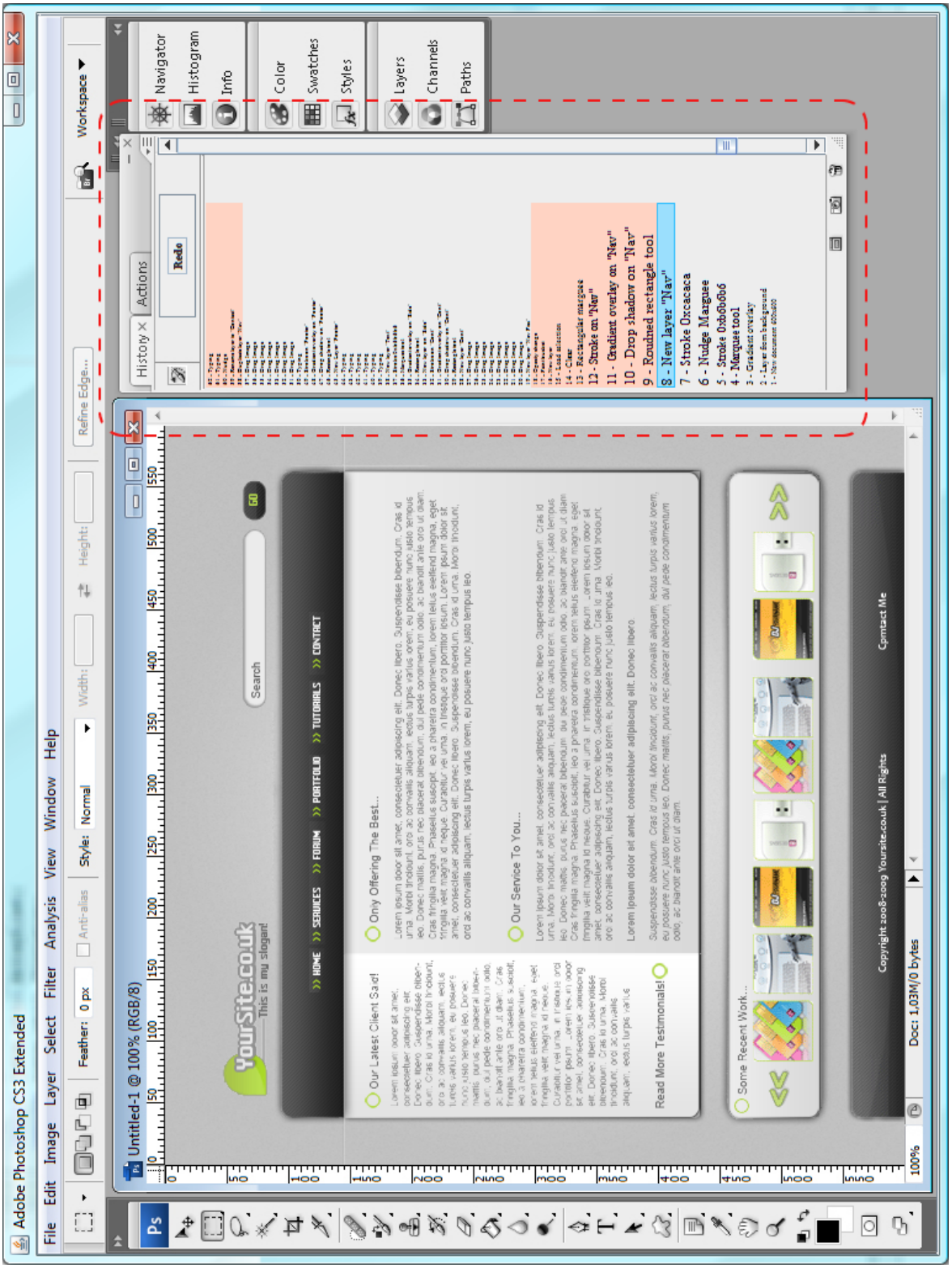


Figure 5.9: Fisheye with scroll with the evaluation environment of Adobe Photoshop CS3, the history list is highlighted.

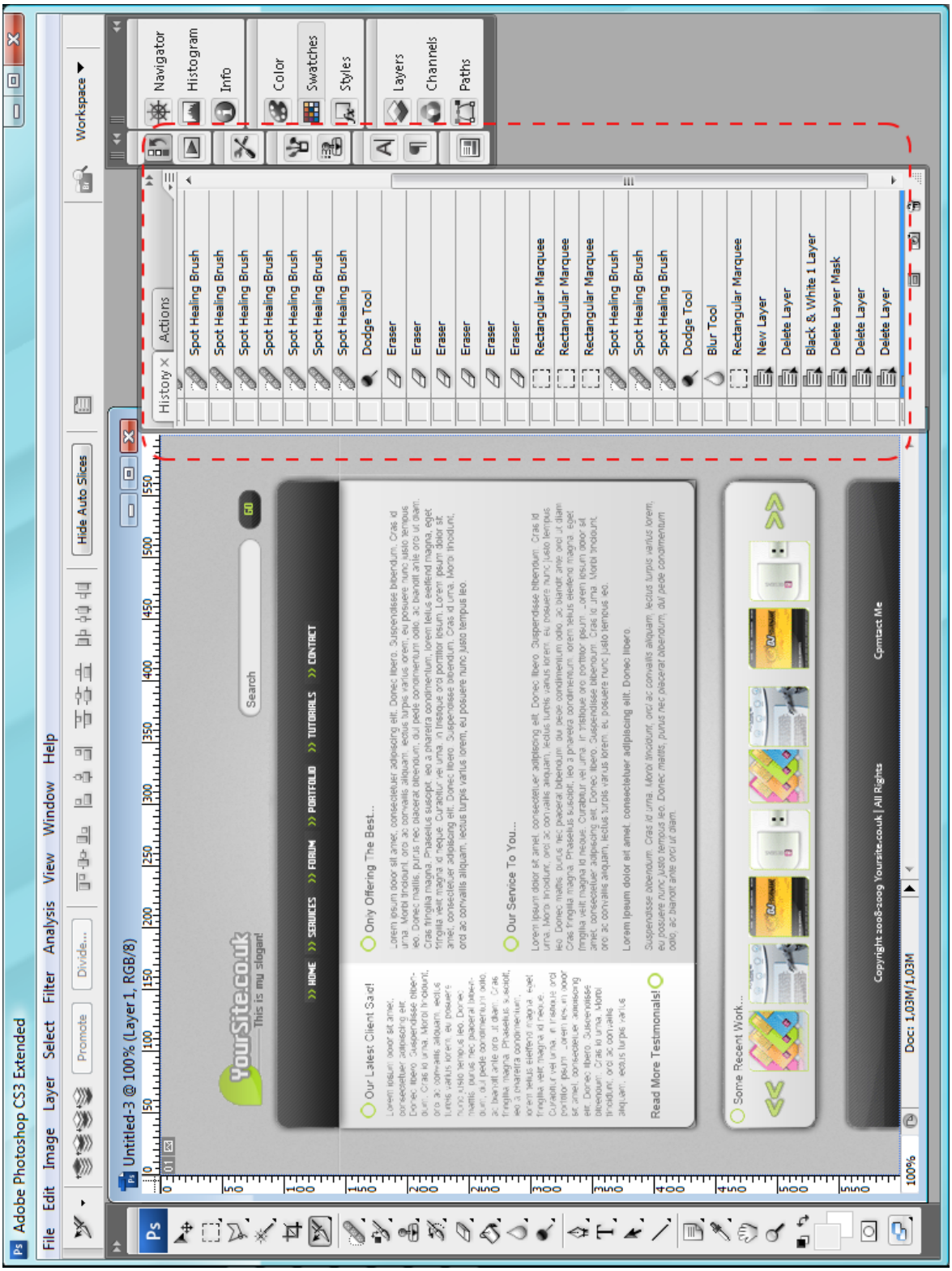


Figure 5.10: Linear history list of Adobe Photoshop CS3, the history list is highlighted.

5.12 Summary

In this chapter we have overviewed implementations of cascading undo visualization. Major issues we encountered during the development were explained and their solution was outlined. The outcome of the implementation will play the key role in evaluations which the next chapter talks about.

Chapter 6

Evaluation

This chapter discusses evaluation of implemented undo visualizations. The results of the evaluation and some assumption are presented.

6.1 Evaluation

6.1.1 Goals

We conducted a pilot study comparing user preference of Simple list, Fisheye with detail and Fisheye with scrollbar visualizations as designed at Chapter 4 and implemented at Chapter 5. The highway was not evaluated since we are convinced that it would not bring any benefits.

The intent of this study was to get preliminary idea of which undo visualization is preferable for users. We do not expect that the results will choose the best visualization. Rather, we hope to get a rough idea of user's preferences that would let us know if our expectation were practical and pragmatic. Future evaluations will, as we hope, continue in what has been done so far.

6.1.2 Set-up

The implementations were placed to the environment of Adobe Photoshop CS3, which was imitated as a set of images. To gain real-user experience we associate the images with the undoes' items. Thus, when a user decided to undo an item in the history list, she could immediately see what the result of the action was. The actions in the history list represented design of a web page.

The study was conducted partially at the Usability Lab of Czech Technical University, Prague, Figure 6.1, and at class rooms. The subjects and their actions were recorded using Camtasia Recorder Studio 5, Figure 6.2.

The testing machine was Dell Inspiron E1505, Intel Core 2 Duo, 2 GB RAM, 1680 x 1050 pixels.

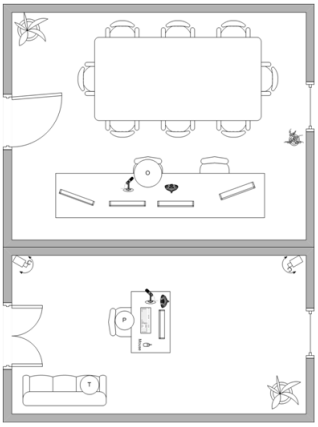


Figure 6.1: Usability Lab

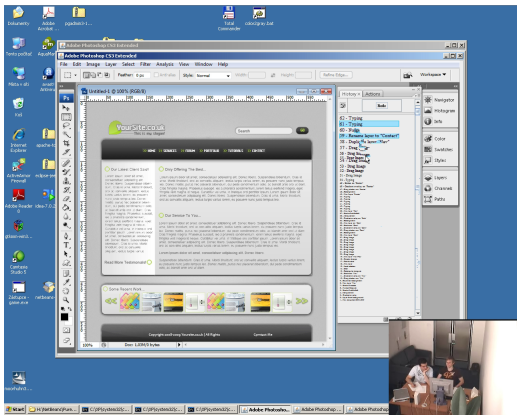


Figure 6.2: Video taken during the evaluation

6.1.3 Subjects' profile

We picked six users, Table 6.1.3 that had at least basic knowledge of the Adobe Photoshop. Three of them had some experience with building web site's design in Photoshop. All users worked with a computer on daily bases and they all use the history tab for comparing the results of some actions with previous state of the document. The subjects were in their 20's.

	Age (years)	Computer usage	Photoshop experience	History list usage
Subject 1	23	daily	basic	comparing, undoing
Subject 2	25	daily	good	undoing
Subject 3	24	daily	good	comparing, undoing
Subject 4	23	daily	advanced	comparing, undoing
Subject 5	21	daily	good	undoing
Subject 6	27	weekly	basic	none

Table 6.1.3: Overview of the evaluated subjects.

6.1.4 Evaluation run

The test started with explaining the purpose of cascading undo. This was shown on a MS Word file containing some elements with dependencies. The disadvantage of the linear undo was presented, and the notion of cascade was introduced. Then the actual structure and process of creating the web site's design were presented with a stress on dependencies among the actions.

The visualizations were presented to users in random order to eliminate the bias in favor of the globaly-first presented visualization. The relations between the principles of the cascading undo were explained at the beginning of the test and the principles of the implementations were demonstrated. The subject was then instructed to try out each

of the three undo types, having as much time as they liked. At this point they were encouraged to ask any questions about how the visualizations work and to talk aloud about what they perceive and think.

Then, the subject was instructed to select an item with non-continuous cascade, and to name all the actions in the item's cascade. Thus, the subject was indirectly instructed to use the means for exploring the cascade. Next task was to remove a group of task without specifying their position in the history list. Thus, we made the subjects to seek in the history list and actively remove actions.

The subject was asked to rate the implementations using 5-point Likert scale and to tell comments on each of them. Finally the subject was asked to rank the three implementations in order of their preference for using the visualizations in real environment.

6.2 Results and analysis

The results were not analyzed statistically, since the study contained a small number of subjects.

Generally there was not a problem with understanding the cascade. When the highlighting appeared every evaluated subject realized that this was the actions which would disappear from the history list if the undo is committed.

All the subjects had difficulties with identifying relation between the steps in the history list and the actual actions representing them. An item named "Drag image" was not enough to identify the impact of undoing the item. Two subjects (2, 4) suggested that the impact of undoing an item should be outline before committing the item to undo, either by highlighting the item's action in the document, or by displaying a preview of the document without the item and its action. One subject (3) proposed that if the mouse cursor enters an element of the document the corresponding item in the history list should be highlighted.

When the subjects were instructed to remove several items in the second task, they complained that the removing several items one by one is annoying. All of them agreed that the multiple-selection feature should be implemented, using ctrl and shift keyboard, to increase productivity.

Two subjects (1,4) did not like that in the fisheye with scrollbar and the simple list implementations the double click had to be carried out in order to commit the undo action. They preferred the one click variant of the fisheye with scrollbar.

6.2.1 Simple list

Five subjects (except 6) reported that they understood the concept of the list since they had already known it. When the cascade is split its parts might go beyond the view port of the history list. The four (1, 2, 5, 6) subjects did not discover the hidden part of the cascade. Three of the four subjects wished to see some sign of this.

The two mode system was well accepted.

6.2.2 Fisheye with detail

At first moment the subjects had difficulties with understanding that the entire history list was displayed. When this was explained all subject got the idea.

One subject (6) reported that there is too much information. The detail view of the cascade seemed not to do its job. Two subjects (1, 6) expressed their dislike of it.

There was confusion about that the split cascade was merged at the detailed view. We believe that the split cascade should be indicated at the detail window, by ellipsis or miniatures inserted between the isolated parts.

6.2.3 Fisheye with scrollbar

The first impression of this implementation was similar to the Fisheye with detail. The fisheye visualization was strange for all subjects at the first moments.

The subjects told that the scrollbar is unnecessary. It did not bring any additional functionality and nobody used it after they explored and played with it.

The two mode system was accepted as in the case of Simple list. One user admitted that he prefers one click undo committing, however another user stated the opposite.

6.3 Summary

We evaluated there implementations with six subjects. The subjects were instructed to carry out tasks that were oriented to find out abilities of the implementation in seeking, committing, and performing the cascading undo.

The table 6.3 plots the result of ranking the implementation on the 5-point Likert scale.

	Simple list	Fisheye with detail	Fisheye with scrollbar
Subject 1	2	3	3
Subject 2	3	4	3
Subject 3	2	4	5
Subject 4	3	5	4
Subject 5	3	4	4
Subject 6	3	2	2

Table 6.3: Subjects' ranking of the visualization on 5-point Likert scale.

Chapter 7

Conclusion

Reverting actions allows exploring and learning software of any kind. Today approach is limiting in this way. We propose new visualizations for progressive cascading undo. We have compared various approaches to visualizations of hierarchical structures with capabilities of displaying large amount of information in small views. The necessary requirements for the visualizations to fulfill our purposes were outlined.

Several visualization principles were selected and combined into new visualizations for cascading undo. The usage scenario described how to control these new visualizations. We discussed pro's and con's of each of them.

Then we implemented four visualizations, Simple list, Fisheye with detail, Fisheye with scrollbar and Highway. The difficulties encountered during the implementation were explained and their solution was provided.

The implementations then were embedded to real software environment, Adobe Photoshop CS3, where the cascading undo might be useful and increase productivity. The evaluations, leveraging these real software imitations, were conducted with Simple list, Fisheye with detail, and Fisheye with scrollbar. During the evaluations we paid attention to various aspects of the implementations, namely, to understanding the relations among the independent action, orienting in the visualizations, and performing actual undoing of the items. The six evaluated subjects answered several question, left their comments and ranked the implementation. The best accepted was the Fisheye with scrollbar, follow by Fisheye with detail and Simple list.

Based on our preliminary evaluation, we believe that the cascading undo approach combined with our visualizations is promising. Clearly, the cascading undo is not suitable for every software, but may enhance usability of various types of programs.

As the future work, the feedback we gathered during the evaluations should be implemented and taken into account. Other evaluations are needed to really find out the real contribution of the cascading undo. The visualizations, we have proposed, should be compared right with the linear undo model in the real software environment. Also the response time of the subjects was not measured, even though it might suggest intuitiveness of the visualizations.

Bibliography

[Bed00] Benjamin B. Bederson. Fisheye menus. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 217–225, New York, NY, USA, 2000. ACM.

[Ber94] Thomas Berlage. A selective undo mechanism for graphical user interfaces based on command objects. *ACM Trans. Comput.-Hum. Interact.*, 1(3):269–294, 1994.

[CF06] Aaron G. Cass and Chris S. T. Fernandes. Using task models for cascading selective undo. 2006.

[CFP06] Aaron G. Cass, Chris S. T. Fernandes, and Andrew Polidore. An empirical evaluation of undo mechanisms. In *NordiCHI '06: Proceedings of the 4th Nordic conference on Human-computer interaction*, pages 19–27, New York, NY, USA, 2006. ACM.

[EK] Jarke J. van Wijk Ernst Kleiberg, Huub van de Wetering. Botanical visualization of huge hierarchies.

[Gut02] Carl Gutwin. Improving focus targeting in interactive fisheye views. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 267–274, New York, NY, USA, 2002. ACM.

[HL06] Stéphane Huot and Eric Lecolinet. Spiralist: a compact visualization technique for one-handed interaction with large lists on mobile devices. In *NordiCHI '06: Proceedings of the 4th Nordic conference on Human-computer interaction*, pages 445–448, New York, NY, USA, 2006. ACM.

[htuu06] http://e-texteditor.com/blog/2006/making-undo usable. A modern undo - making undo usable beyond the last few changes. Blog, 2006. The author remain unknown.

[JEACS84] Jr. James E. Archer, Richard Conway, and Fred B. Schneider. User recovery and reversal in interactive systems. *ACM Trans. Program. Lang. Syst.*, 6(1):1–19, 1984.

[LR94] John Lamping and Ramana Rao. Laying out and visualizing large trees using a hyperbolic space. In *UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 13–14, New York, NY, USA, 1994. ACM.

[M.J] M.Ranlöf A. Nilsson* M.Jern, S. Palmberg. Coordinated views in dynamic interactive documents.

[MRC91] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. The perspective wall: detail and context smoothly integrated. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 173–176, New York, NY, USA, 1991. ACM.

[RM93] George G. Robertson and Jock D. Mackinlay. The document lens. In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 101–108, New York, NY, USA, 1993. ACM.

[RMC91] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone trees: animated 3d visualizations of hierarchical information. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 189–194, New York, NY, USA, 1991. ACM.

Appendix A

Content of the Enclosed CD and User Manual

A.1 Content of the enclosed CD

The content of the enclosed CD is organized into two main directories.

- **\thesis**
  contains the pdf version of this thesis
- **\implementations**
  includes the Net Beans projects and the compiled .jar files of the implementations

A.2 User manual

The implementations are started as listed:

- Simple list via
  \implementations\SimpleListUndo\SimpleListUndo.bat
- Fisheye with detail via
  \implementations\FisheyeDetail\FishEyeDetail.bat
- Fisheye with scrollbar via
  \implementations\FisheyeScrollbar\FisheyeScrollbar.bat
- Highway via
  \implementations\Highway\Highway.bat

A.2.1 Control

The only input option is the mouse. If you move the mouse over the history list part, the visualization react according its proposal, Chapter 4. If a double clicked is issued the undo carries out its actions and the Photoshop project changes accordingly.

Use the *Redo* button at the upper part of the history list to redo the undo actions.

Appendix B

Evaluation materials

This appendix presents some of the materials used during the evaluation.

B.1 Pre-test questionnaire

The pre-test questionnaire, on Figure B.1, aimed to find out relevant information about the subjects.

B.2 Post-test questionnaire

The post-test questionnaire, on Figure B.2, aimed to rank the evaluations and get subjects' feedback.

B.3 Document structure

The document structure, on Figure B.3, provided an overview of how the parts of the Photoshop project are related. Thus, the subjects got an initial idea about the document.

Pre-test dotazník

1. Jak často pracujete na PC?	• denně • týdně • měsíčné
2. Používáte zpět nebo raději editujete?	• zpět • edituji
3. Jaká je Vaše znalost Photoshopu?	• malá (zvládám jen základní úkony) • střední (udělám, co potřebuji) • velká (nemám s ničím problém)
4. Jak často používáte "záložku Historie" ve Photoshopu?	• často • občas • vůbec
5. Popište stručně situaci, kdy používáte záložku „Historie"	• porovnání předešlého a současného stavu • zrušení posledních kroků • revize doposud provedených kroků
6. Máte zkušenosti s návrhem webových stránek ve Photoshopu?	• ano • částečně • ne

Figure B.1: The pre-test questionnaire

Post-test dotazník

1. Jak se Vám líbí List Undo?	Líbí Nelíbí 1-------2-------3-------4-------5
2. Bylo vždy zřetelné, co bude z dokumentu odebráno?	• ano • ne
Komentář:	
3. Jak se Vám líbí FishEye Undo s detailním pohledem na pravé straně?	Líbí Nelíbí 1-------2-------3-------4-------5
4. Bylo vždy zřetelné, co bude z dokumentu odebráno?	• ano • ne
Komentář:	
5. Jak se Vám líbí FishEye se scrollbarem?	Líbí Nelíbí 1-------2-------3-------4-------5
6. Bylo vždy zřetelné, co bude z dokumentu odebráno?	• ano • ne
Komentář:	

Figure B.2: The post-test questionnaire

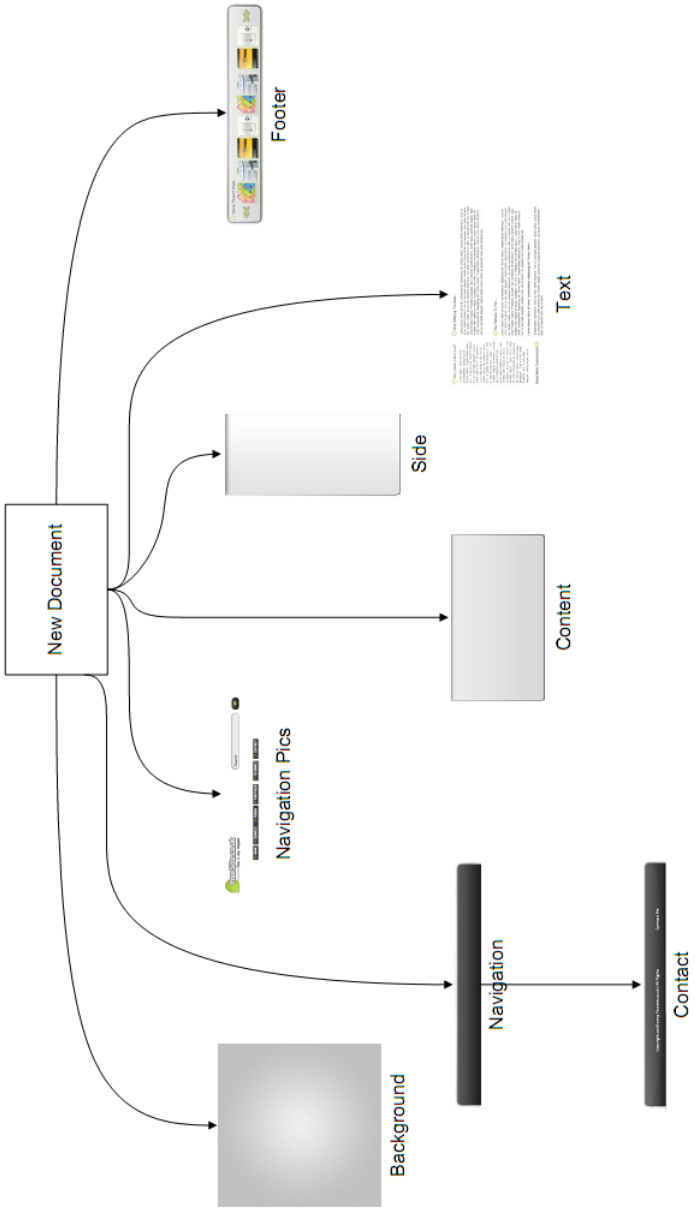


Figure B.3: The structure of the document