Creating a Visual XML Editor

Ivar Ekman

University of Tampere
Department of Computer Sciences
Interactive Technology
Master's Thesis
Supervisor: Professor Kari-Jouko Räihä
May 2008
XML files are widely used but few generic visual editors exist. Most editors only display the content as text with color coding and indentation. In this thesis existing XML editors are evaluated and, as XML files are trees, several existing visualizations for hierarchies are presented in order to find appropriate ones for presenting XML files. Improvements are suggested for Icicle plot and Cheops but otherwise left for future research. A new tree visualization technique, Tabbed Treemaps, is developed and a new XML editor based on it is created and evaluated. With XML documents up to 100 kilobytes the new editor is found to give a compact overview of the document in a way not possible in other editors. In order to work well with larger files and develop the editor in general, several improvements are suggested but left for further research.

Keywords and phrases: XML, visualizing, tree, Treemaps.
Contents

1. Introduction .................................................................................................................. 1
2. XML file used for testing ................................................................................................. 3
3. Visualization objectives .................................................................................................. 6
   3.1. Showing structure ........................................................................................................ 6
   3.2. Showing content ............................................................................................................ 8
4. Existing domain independent XML editors ................................................................. 10
   4.1. Evaluated editors ......................................................................................................... 10
      4.1.1. Eclipse Web Tools Platform XML Editor .......................................................... 10
      4.1.2. Microsoft XML Notepad 2007 ........................................................................... 12
      4.1.3. Oxygen XML Editor .......................................................................................... 14
   4.2. Results of the evaluation ............................................................................................. 19
   4.3. Possible improvements ................................................................................................. 23
5. Existing tree visualizations and how they can be used with XML ...................... 24
   5.1. Text ............................................................................................................................ 24
   5.2. Node-link diagrams ................................................................................................... 24
   5.3. Cheops, radial presentations and Icicle plot ............................................................. 27
      5.3.1. Suggested improvement: Focus+context Icicle plot ........................................... 30
      5.3.2. Suggested improvement: Cheops from the upper left corner and combining visualizations .......................................................... 32
   5.4. Nested circles ............................................................................................................. 33
   5.5. Summary of the evaluations of tree visualization ................................................. 33
6. Treemaps ......................................................................................................................... 37
   6.1. Existing Treemaps visualizations ............................................................................. 37
   6.2. Tabbed Treemaps ...................................................................................................... 41
7. A new XML editor ........................................................................................................ 45
8. Evaluation of editing XML documents and the new XML editor ..................... 51
   8.1. Objectives ................................................................................................................. 51
   8.2. The evaluation process ............................................................................................... 51
   8.3. Editing process in the test group .............................................................................. 52
      8.3.1. Edited documents ............................................................................................... 52
      8.3.2. Actions performed with the XML documents .................................................... 53
      8.3.3. Used editors and features .................................................................................. 54
   8.4. Evaluation of the new editor ...................................................................................... 55
      8.4.1. Visualization ....................................................................................................... 55
      8.4.2. Navigation .......................................................................................................... 59
      8.4.3. Editing ................................................................................................................ 61
      8.4.4. Other features .................................................................................................... 62
8.5 Discussion on the suggested improvements ............................................. 62
  8.5.1 Visualization .............................................................................. 62
  8.5.2 Navigation ............................................................................... 67
  8.5.3 Editing ..................................................................................... 68
  8.5.4 Other features .......................................................................... 70
8.6 Summary of the evaluation .................................................................... 70
9 Conclusion .............................................................................................. 72
References ............................................................................................... 74
1. Introduction

XML files [Bray et al., 1998] can be used to store almost any data. Both Microsoft Office and OpenOffice.org are beginning to move towards using, or already use XML file formats. HTML, RSS and many other files used on the Internet are XML files. Even games and programs can be created with XML (MUPE [Mupe, 2006] and VoiceXML [McGlashan et al., 2004], for example). XML has been successful because it allows saving data in a human-readable way. The definition of the data and the data itself are stored using tags, attributes and data inside these tags or attributes. An XML file can be opened, read and edited with a text editor.

The binary format is completely different from XML. Data is stored in a computer-readable way. A binary file can be such that only the creator knows how it is constructed. Therefore only a program made for editing the file can open it. XML files, on the contrary, can always be read and edited as they are text files with tags that have a special meaning.

Although XML files can be opened with any text editor, they can be so large and complex that they are hard to handle in text editors. This means that the XML file format makes it theoretically possible to open and edit a file without a domain specific program. In practice, however, one needs the correct program for some files. If no domain specific editor exists, the only possible editor is a generic XML editor. If a text editor is used, it complicates editing files, as there are no advanced tools like in domain specific editors and the file is displayed as it is without any abstractions or graphical representations.

Current domain independent XML editors mostly show the code, possibly coloring tags and using indentation to show the structure of the file, Oxygen [Oxygen, 2008] probably being the most advanced one with several graphical representations. If the file is long and complex, editing becomes hard even for advanced users. A user without knowledge on how XML works will probably not be able to edit the file at all.

A node-link diagram similar to the one in Windows Explorer is the most common tree visualization used in existing XML editors. In general, a node-link diagram use some representation for nodes, such as circles, with node names in them, and lines to connect these nodes to show the structure. In Windows Explorer nodes are shown on their own lines and indentation is used instead of lines to show the structure.

I argue that it is possible to visualize the structure of XML files and use advanced tools for editing them. It is possible to adopt some tools from domain dependent editors (bound to a specific type of XML) and use them in a domain independent XML editor. These tools can be buttons, menus, text fields or anything one already is familiar with in other programs. It is also possible to display the tree-structure of an XML file in various
ways. In addition to indentation and the node-link diagram other tree visualization

techniques, such as Treemaps, can be used.

This thesis studies visualizing XML in different ways as a part of creating a domain

independent visual XML editor based on the Treemaps visualization. A novel

visualization, Tabbed Treemaps, is presented as part of this. It uses Nested Treemaps for

showing the structure and adds tabs to the rectangles in order to show the names of the

nodes.

In order to test existing editors and evaluate different XML visualizations a test

XML document is created and described in Chapter 2. Chapter 3 defines some research

questions and visualization objectives one has to answer when creating a new XML

editor.

XML editors that visualize XML are evaluated in Chapter 4. In Chapter 5 existing

tree visualization techniques are presented with discussion on how they can be used for

visualizing XML. Tabbed Treemaps are presented in Chapter 6 and the implementation

of an XML editor based on them in Chapter 7.

Chapter 8 describes how the editor has been evaluated and discusses how it can be

improved. Chapter 9 summarizes the results and gives directions for further research.
2. XML file used for testing

XML files consist of tags, attributes, text, CDATA sections, comments, processing instructions and document type declarations. A tag forms a level in the tree structure that can have text, other tags, comments and processing instructions inside it. Attributes are name-value pairs used to define properties for tags. Text and CDATA elements contain the main content of the document. The difference between them is in the notation of special characters. Comments can be used to annotate on the document. Processing instructions are directions for applications. They consist of one target with several name-value pairs. A processing instruction in the beginning of the document is called the XML declaration and specifies which version of XML is used. [Bray et al., 1998]

In order to discuss various XML visualizations, a test XML document has been created that contains all possible elements in an XML file, except for a document type definition. It is ignored throughout this thesis as I classify it being a separate document. The test XML contains a wide and a deep structure. This file is used when discussing research questions in Chapter 3 and for presenting existing editors in Chapter 4 and the new editor in Chapter 6.

The test XML document is presented in Figure 1. It contains different type of elements within their own tags, in tags 1 to 8. Tag9 contains the same elements within one tag and Tag10 presents a special case with text with a tag in the middle of it. Tag11 contains a high tree with ten levels. Tag12 has three wide trees with 10, 100 and 1000 children. In Tag12.1, Tag12.2 and Tag12.3 most of the children are not written here to keep this presentation short.

```xml
<?xml version="1.0" encoding="utf-8"?>
<roottag>
    <Tag1></Tag1>
    <Tag2><Tag2.1_leaf /></Tag2>
    <Tag3 Tag3AttributeName="Tag3AttributeValue" />
    <Tag4>Tag4Text</Tag4>
    <Tag5><![CDATA[Tag5CDATA]]></Tag5>
    <Tag6><!-- Tag6Comment --></Tag6>
    <Tag7><?Tag7ProcessingInstruction?></Tag7>
    <Tag8><?Tag8ProcessingInstruction Tag8ProcessingInstructionName="Tag8ProcessingInstructionValue" ?></Tag8>

    <Tag9 Tag9AttributeName1= "Tag9AttributeValue1" Tag9AttributeName2="Tag9AttributeValue2">
```
<Tag9.1_leaf />
Tag9Text
<![CDATA[Tag9CDATA]]>
<!-- Tag9Comment -->
<?Tag9ProcessingInstruction Tag9ProcessingInstructionName1="Tag9ProcessingInstructionValue1" Tag9ProcessingInstructionName2="Tag9ProcessingInstructionValue2" ?>
</Tag9>

<Tag10>Tag10.2Text1<Tag10.2 />Tag10.2Text2</Tag10>

<Tag11>
<Tag11.Level1>
<Tag11.Level2>
<Tag11.Level3>
<Tag11.Level4>
<Tag11.Level5>
<Tag11.Level6>
<Tag11.Level7>
<Tag11.Level8>
<Tag11.Level9>
   Tag11Level9Text
</Tag11.Level9>
</Tag11.Level8>
</Tag11.Level7>
</Tag11.Level6>
</Tag11.Level5>
</Tag11.Level4>
</Tag11.Level3>
</Tag11.Level2>
</Tag11.Level1>
</Tag11>

<Tag12>
<Tag12.1>
   Tag12.1.1 /
   Tag12.1.2 /
   ..

</Tag12>
Figure 1. Test XML document used throughout this thesis.

Figure 1 is one way to present XML documents. Indentation is used to enhance the structure. Even with a document this small, it is hard to apprehend the structure of the document. For example, without descriptive tag names it is hard to know how many siblings Tag12 has and how much of the document is in its subtree. The larger and more complex an XML document is, the harder it becomes to navigate and edit it without advanced visualizations.
3. Visualization objectives

There are several questions that need to be answered when visualizing an XML file. As the editor is to be domain independent, not many assumptions can be made. The XML file can be of any size and structured in any way as long as it follows XML regulations. Next the main questions for this thesis are defined, the parts that build up a visualization of XML.

When discussing XML files and trees in general, their structure will be referred to using the terms parent, grandparent, child and sibling. A parent is a node that includes another node. The included node is a child of the parent. A grandparent is a parent higher up in the hierarchy than the parent. Nodes on the same level are siblings.

3.1. Showing structure

XML files are used because of the tree structure. Without the need for a structure a plain text file could be used instead. An XML file is always a tree, with one root. Everything within the document is within the root tag (with the exception of the document declaration), usually within several layers of tags. All tags form their own subtrees. The number of tags in the document or within another tag is not limited nor is the height of the tree.

My experience has shown that there are seldom more than seven levels of tags in XML files, but nothing restricts this. It seems that XML trees are usually more wide than high. One tag has several children in the same layer instead of several layers of children, which seems to be what makes an XML file big. This, however, cannot be taken for granted as this information is not based on any published research.

Showing the tree structure can be divided into several subquestions. These will now be examined with discussion on how they can be answered. Answers to some of these questions are found in existing tree visualizations, but other questions are XML specific.

Parent or child

Showing the structure starts from showing that a node is a parent. In a node-link diagram this can be shown using a special icon in front of the tag name to show that the tag has children. The icon can be a plus or minus sign or a closed or open folder to show if the children are shown or not. Color can also be used for showing the difference between a parent and a child.

Structure is often indicated using relative position. For example, in text indentation of a line under another line shows that the node on the upper line is a parent. In Figure 1 the structure of Tag11 is shown this way. One can also use textual notation, such as numbers or parentheses, to show the structure. This is demonstrated in Figure 10.
Node-link diagrams use lineation and positioning to show the relation between nodes. For example, a parent can be positioned above its child and connected to it with a line. A tree structure can also be shown with nodes inside or outside each other. That a node is positioned within another node can thus mean that it is a parent or a child.

**Root node**
Showing where the tree starts from is often implicitly shown by the structure, but it can also be necessary to show this by differentiating the root from other nodes. This is especially important if the used visualization is unknown to the user or if the structure is such that it is impossible to know which node is the root.

**Leaf tags (does / does not have children)**
In XML files, text is always at the end of the tree structure, in the leaf tags. Alternatively text nodes themselves can be classified as leaf nodes. Some visualizations only show the leaf nodes, whereas others highlight them in some way.

**Subtree size (width and height)**
With large XML files it can be necessary to show how many children a tag contains. As will be seen in Chapter 4, several existing editors provide a possibility to hide parts of the structure. Especially in this case it might be necessary to show how much content is hidden. In a node-link diagram this can be shown by the number of lines leaving a node. SpaceTree [Plaisant et al., 2002], discussed in Chapter 5, shows the subtree as a triangle. The number of children is shown by shading, the height of the subtree by the height of the triangle and the width with the base of the triangle. The presentation of the node, for example its size, can also be dependent on the number of nodes in the next level(s). Instead of graphics, this information could also be shown using numbers.

**Number of siblings**
When editing one of several children, it can be advantageous to show where the current child is related to its siblings. This is important especially if some of the siblings are hidden. This can be shown using shading within the siblings, for example, so that the first one is light and the last one dark. This can also be shown with numbers.

**Tag level (height)**
In text or in a node-link diagram indentation can show the number of levels. In Nested Treemaps [Johnson and Shneiderman, 1991] (see Chapter 6) this is shown with the number of rectangles around the selected tag. The same information can also be shown using the path to the tag. If this is not shown directly it has to be solved by counting levels.
Path to tag
Solving the path to a tag can be a difficult task. If parts of the path are hidden one needs to navigate the tree to solve the whole path. As discussed in Chapter 4, some editors, like Oxygen XML editor [Oxygen, 2008], ease solving the path by showing the path in a separate part of the window. In other editors this has to be solved by navigating to the node or to the root.

3.2. Showing content
In addition to the structure, text and attributes are the main content of an XML document. Text can be within a CDATA section and thus contain any character sequences except “]]>”, as this ends the CDATA section [Bray et al., 1998]. Special characters, such as the ampersand sign are not escaped in CDATA sections. When showing text, there is one special situation to consider. The structure can be such that a tag has text, then a tag with a subtree, and then text again. This situation is demonstrated in Tag10 in the test XML file in Figure 1. The tag in the middle of the text, Tag10.2, could be a large subtree. How should this be shown? Text editors show the subtree in its entirety but if the subtree is of minor importance, it could be hidden instead.

A comment is restricted by less strict rules than text elements. It can contain any text except for a double-hyphen “--”. This is because the characters “-->” are used to end the comment [Bray et al., 1998]. Comments can be free text describing the document but also a disabled part of the structure. Processing instructions are instructions for applications.

Next, I will discuss how XML elements and their content can be presented.

Tags
Showing a node in a tree can be done in several ways. This will be presented in Chapter 5 in greater detail. In short, elements can be shown as any shape, such as circles and rectangles. To differentiate elements from each other, one can use a wide range of options, including borders, color, shading, icons in front of names and textual icons.

Text and CDATA
Text can be either text by itself or within a CDATA section. Text within a CDATA section has fewer restrictions on permitted characters (necessity to escape characters). Thus, CDATA sections can be shown differently than text, or identically to it, if text entities are decoded.

Attributes
Attributes are used to define properties for tags by names and values attached to the names. This resembles leaf tags with text inside them. Attributes can thus be shown
using the same means as those described for showing tags with text inside them. In text like in Figure 1 and several existing editors presented in Chapter 4, attributes are shown as text beside the tag name.

**Comments**

Comments are often handled as plain text, but they can also contain a disabled part of the XML document with structure. The decision is thus to show them as text (XML code) or visualize them like tags, but differentiate commented tags from normal tags somehow. One option for doing this is to gray out commented tags. However, comments can also contain invalid structures (not permitted by XML). These cannot be shown as tags, but have to be shown as text.

**Processing instructions**

Processing instructions are similar to tags with attributes but they cannot contain children. They could thus be shown using the same means as tags and attributes.

Next the questions defined in this chapter are used when evaluating existing XML editors and tree visualizations. In Chapter 4, I will evaluate existing XML editors in order to find out how current editors present XML documents. XML files are trees and existing tree visualizations can be used at least for showing the structure of XML documents. Thus, in Chapter 5 I will discuss how different tree visualizations can be used in a visual XML editor.
4. Existing domain independent XML editors

As XML files are text, it is possible to edit them in any text editor, but there are also several editors made especially for editing XML files. Editors made for specific types of XML files work well for them, but badly or not at all for unsupported files. Domain independent editors, on the other hand, work with all XML files but not optimally. Next I will give a brief overview of available editors that visualize XML in some way. I will evaluate them in order to find important features and ideas for creating a new editor.

I have only included editors with distinctive visualization features. I have not evaluated XML editors that only visualize XML using syntax highlighting and tag expanding / collapsing as these features are available in almost all editors, including most of those now evaluated. Well-known editors not discussed here in detail include KatePart [Kate, 2007], Jedit [Jedit, 2007], Crimson Editor [Crimson, 2008], and Altova XMLSpy 2008 [XMLSpy, 2008]. KatePart is an editor component used in various KDE Linux applications, such as Kate. It has various highlight schemes for different file types, including domain independent XML files. Jedit is a Java-based application that thus works on any platform that supports Java, with similar text highlighting features. Crimson Editor is a similar program to Kate and Jedit but works only on the Windows platform. Altova XMLSpy 2008 is an editor with similar features to Oxygen XML Editor [Oxygen, 2008], but I have decided to evaluate Oxygen XML Editor instead as it has more visualizations.

4.1. Evaluated editors

4.1.1. Eclipse Web Tools Platform XML Editor

Eclipse Web Tools Platform (version 2.0.1) [Eclipse, 2008] has a built-in XML editor with some XML visualizing features in addition to an XML text editor. It has three views for XML files: the outline view, the design view, and the source view, in addition to a properties view for tags. The outline view and the design view are similar and resemble a node-link diagram, and the source view is a text editor. The views are updated simultaneously – selecting a node in one view makes it selected in the other views. One can choose to show either the source view or the design view. The other views are shown separately in their own windows and can be shown or hidden. Figure 2 shows the outline and source views.
The source view is a text editor with syntax highlighting and a darker bar on the left showing how far the current tag or comment spans. Tags, attribute names, attribute values, comments and text are shown in different color. The tree structure is (on request) shown using indentation. As the source view is a text editor it can also edit invalid XML documents, and validation errors are shown by marking the invalid parts in addition to listing the errors in a separate problems view.

The outline view visualizes the XML as a vertical expandable node-link diagram with the structure of the document. The level of an element is shown with indentation. Tags, CDATA sections, comments, and declarations have their own icons. Attribute names with their values are shown after the tags but text and comments are not shown, only their placeholders.

The design view, shown in Figure 3, is similar to the outline view but shows more information and uses two columns. The left column resembles the outline view and tags, CDATA sections, comments, and declarations have the same icons as in the outline view but without the text “#cdata-section” and “#comment” as the outline view has. In addition to this, attributes are shown as children using an “a” in a circle as icon with the attribute name next to the icon. If text is in a tag with other elements, such as attributes or tags, the place for the text is shown using an icon that looks like a paper sheet.

Text, CDATA content, attribute values, comments and declaration contents are shown in the right column. This way the structure is separated from the content.
The properties view is available always when a tag is selected and lists its attributes in a two-column node-link diagram. The attribute names are in the left column and the values on the right.

Searching is possible only in the source view and is based on a normal text search with case sensitivity and regular expressions. It is also possible to search and replace text without restrictions on the XML file. In the outline and design views it is possible to change the order of items by dragging and dropping. Adding items is possible from a menu shown in Figure 3 or by copy-pasting. In the source view everything is text, and normal text editing functions are available.

4.1.2. **Microsoft XML Notepad 2007**

Microsoft XML Notepad 2007 (version 2.5.2798.17141) [Xmlnotepad, 2007] is an XML editor that only can open valid XML documents. It separates the structure and content into two separate sections. Figure 4 shows the XML file presented in Figure 1 opened in the editor. On the left the structure is shown with different icons for tags with children, leaf tags, attribute names, comments, CDATA sections and processing instructions. On the right of these placeholders the values are shown. This way the structure is shown on the left and the content on the right. If the content does not fit on the line reserved for it, the content is cut and an ellipsis shows that more content exists.
Tags with content inside them (text or children) have folder-icons in front of them. Leaf tags have blue dots instead of folders with the special case of a tag with only text inside it. When collapsed the icon is a blue dot and when expanded the icon is a folder. In Figure 4 the child of Tag11.Level8, Tag11.Level9, is collapsed and thus has a blue dot. When expanded a placeholder for the text element is added as a child to Tag11.Level9 and the icon for the tag changes to an open folder.

Text and CDATA sections use the same icon, a notepad, in the right column except that for CDATA sections the icon is gray. In addition to an icon the text “#text” or “#cdata-section” is used to present the element type. On the right the content is shown in black for text and gray for CDATA sections. If a tag is not expanded and only has text inside it, as in Figure 4 for the Tag11.Level9, the text is shown right to the tag and in blue.

Attributes are indicated by red dots as their icons and are shown in red text. Processing instructions use purple dots and purple text. Attributes show the name in the left column and the value in the right whereas processing instructions show the target in the left column and the names and values in the right column. Comments use green dots and the text “#comment” in the left column and the content is shown in green text in the right column.
Content can be edited by selecting the wanted element, for example, a text section that expands for the time of editing. The declaration is hidden and encoded characters are shown as decoded. If one needs to edit an invalid document, Microsoft XML Notepad suggests opening it in Notepad. The same option is also the only possibility for showing the XML document as text without any visualization.

In the left section it is possible to drag and drop elements and add and remove elements by right-clicking and copy-pasting. In the right section the menu and functions are the same, but dragging and dropping is not possible. There is an advanced search with case sensitivity, regular expression search and an option to search within tags, text, comments or everything.

4.1.3. Oxygen XML Editor

Oxygen XML editor version 9.1 [Oxygen, 2008] is a commercial editor with several XML visualizations. Its main views are a text editor, a grid view, an author view, an outline view and a tree editor. The tree editor opens in a separate window and does not interact with the other views. The outline view is available in its own section of the screen and works together with the text view and the author view. There is also a model view and an elements view to show additional information for the document, possibly defined in a DTD or schema. The additional views are presented in Figure 5. Otherwise the text view, the grid view and the author view work as the main view to the XML file. Only one of them can be open at a time.

The outline view is a node-link diagram with blue dots in front of tag names. If the tag is expanded the blue dot looks pressed down in the middle. Attribute values are shown in gray text within quotation marks next to the tag name. Attribute names and CDATA sections are not shown. Processing instructions are shown as they appear in XML as child elements in gray under their parents. Subtrees can be expanded and collapsed using + and – icons next to them if they have children.
Figure 5. Oxygen XML Editor: Outline, Model and Elements views.

The text view is a text editor with syntax highlighting and tag collapsing. A tag that spans several lines can be collapsed and expanded by clicking on an arrow left to it. An arrow pointing downwards shows that the subtree is shown. When the arrow points to the left the subtree is hidden and the number of hidden lines is reported within square brackets next to the tag. Unlike Eclipse and several other text editors, Oxygen XML editor has a context-sensitive menu shown in Figure 6 that is accessible by right-clicking the document. A tag can be surrounded by comments and the prefix or name of a tag can be edited globally or within the subtree of a selected tag using this menu. This is also the only view within the Oxygen XML editor where it is possible to edit invalid XML documents.

Figure 6. Oxygen XML Editor: Text view and the context-sensitive menu.

The grid view consists of nested tables where the right cell in the table contains a table representing the subtree of the node on the left. If there is no child tree, the cell contains a possible element definition and a value for the element. Text is shown in one column if it is the only element in the tag, otherwise using two columns, one for showing that the element is a text element with the notation “#text” and the other for showing the content. All other elements are shown using two columns. Subtrees can be
collapsed by clicking arrows that work similarly to those in the text view, except for that the number of hidden information is not shown.

CDATA sections have the text “CDATA” in the left column and the content in the right one. Attributes show the name of the attribute with a prefix “@” in the left column and the content in the right one. Comments show “<!--” in the left column and the content in the right one. Processing instructions show the target with the prefix “<?” in the right column and the rest of the content in the right column.

Figure 7 shows the test XML in Figure 1 opened in the grid view. Tag2 is expanded and has one empty tag Tag2.1_leaf inside it. Note that there is no text inside Tag2.1_leaf as can be seen in the empty cell right of it. Tag10 is collapsed and thus does not show its children or properties. Tag9 has several properties and this is shown in the cells right of it. Notice the different presentation of text elements for Tag4 and Tag9. Text within a tag solely with the text element in it is shown as the property of the tag. This is seen in Tag4 in Figure 7. When the tag has other elements, such as attributes in it, text is shown using two columns, as the text in Tag9.

With higher trees the grid view expands to the right and with wider trees it expands downwards. This way a rectangle with the height of the left cell spanning all the way to the right represents the whole subtree. Parts of the subtree can be collapsed to hide the subtree from one point forward. After selecting an element, its content can be edited as text. Dragging and dropping is available as is also changing the structure by right-clicking and selecting functions from a menu. The grid view enables editing the values of leaf text and attribute elements in one place, on the right in the table. It also expands to the right instead of downwards as text editors.
The author view shown in Figure 8 resembles a normal text view but visualizes the tags as boxes with arrows to indicate that it is a starting or closing tag. Indentation is not used consistently to show the structure. Instead a breadcrumb trail is used to show the path to the current tag. This enables editing the content without using space to show the structure. Attributes, comments and processing instructions are hidden but attributes can be edited using a menu. CDATA sections are shown within boxes with the text “#cdata”. With an associated CSS style sheet this becomes a WYSIWYG editor and text can be presented differently. As in Eclipse, a dark bar on the left shows how far the selected tag spans.

**Figure 7. Oxygen XML Editor: Grid view.**
The tree editor is a node-link diagram that expands to the right and uses different icons for tags with children, leaf tags, attributes, comments and processing instructions. The icon for a tag is a node-link diagram expanding to the right. Attributes have two red boxes on top of each other as their icon. Text is shown by a capital T and a box surrounding the text and CDATA in the same way but with a green star as the icon. Processing instructions are shown as they are presented in a text editor using black text.

The tree editor provides a possibility to hide attributes and whitespace to simplify the interface. The path to the current element is shown as a breadcrumb trail on top of the view. Elements can be moved by drag and drop and new elements can be added using a menu or buttons. Double-clicking an element starts editing it. Figure 9 shows the XML file presented in Figure 1 starting from Tag9.

It seems that Oxygen XML editor’s different views are targeted at editing different aspects of the XML file. The tree editor is for altering the structure while the author view is only for editing text. The grid view works well for editing both content and structure, as does the text view also. The grid view is the most exotic visualization and also the most interesting one. It enables editing the structure and the content in a compact way using a visualization not present in other editors.
4.2. Results of the evaluation

Most domain independent XML editors do not visualize XML in any way and even the ones that do usually use graphical presentations only as alternative views. The most common visualization is a node-link diagram with collapsing sometimes integrated into the text. This has, in some editors, been further developed to present the structure and the content separately. Most evaluated visualizations do not use screen space efficiently. For example, the evaluated node-link diagrams have much empty space left to the tree. No evaluated editors include a visualization to get an overview of a large document.

There are several useful tools in the evaluated editors, many of which are only for technologies related to XML, such as Xpath, XSD and XSLT. These have on purpose been left outside of this research. For domain independent XML files it is possible to alter the structure by drag and drop, copying and pasting and using context sensitive menus. Oxygen XML editor enables adding and removing comments around elements and thus disabling or enabling parts of the tree. In the other editors comments are treated as text. Microsoft XML Notepad 2007 has an advanced search functionality that enables searching within a specified element type. Oxygen XML editor enables hiding elements by their type in the tree editor and Eclipse automatically hides information depending on the view. Collapsing nodes is available in many different views but collapsing several nodes in between the nodes one wants to edit can be a lengthy process.
Oxygen XML editor creates a model for the XML file by the existing structure if no DTD or schema is defined. This way it is possible to add elements from a menu with available types based on the existing document. In other editors the name of the node has to be given via typing. No editor uses other advanced tools for editing data, such as radio buttons or sliders, even if a schema or DTD is present.

Table 1 shows a summary of how structure is shown in the evaluated editors. As is seen in the table, most visualizations use similar techniques for showing the structure. These visualizations only represent a fraction of the tree visualizations discussed in Chapter 5. All visualizations show the XML progressing horizontally starting from the left or vertically starting from the top.

The ways for showing content is summarized in Table 2. Elements are differentiated from each other mostly using icons, textual placeholders and color. In several visualizations element positions are shown in one place using placeholders, and the content in a separate area, mostly in another column. Some visualizations do not show the content for all elements or even the elements at all. For example, Oxygen XML editor’s author view only show tags and text elements. This simplifies the visualization, but makes it suitable only for documents where the hidden elements are of no interest.

<table>
<thead>
<tr>
<th>Parent or child</th>
<th>Eclipse source</th>
<th>Eclipse outline</th>
<th>Eclipse design</th>
<th>Microsoft Notepad</th>
<th>Oxygen XML Outline</th>
<th>Oxygen XML text</th>
<th>Oxygen XML grid</th>
<th>Oxygen XML author</th>
<th>Oxygen XML Tree editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Textual notation</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Different icon for other elements than tags</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lines connecting</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Child inside</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Separate path</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Root node</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Lines entering/exiting</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Whole tree inside root</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leaf nodes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Textual notation (&lt;tag /&gt;)</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Missing expansion sign</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Differentiated icon for other elements than tags</td>
<td>Eclipse source</td>
<td>Eclipse outline</td>
<td>Eclipse design</td>
<td>Microsoft Notepad</td>
<td>Oxygen XML Outline</td>
<td>Oxygen XML text</td>
<td>Oxygen XML grid</td>
<td>Oxygen XML author</td>
<td>Oxygen XML Tree editor</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>------------------</td>
<td>--------------------</td>
<td>-----------------</td>
<td>----------------</td>
<td>-------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Position</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Subtree size</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spanning shown as border</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>One element per row/column</td>
<td>R</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Numerical notation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Element size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Number of siblings</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calculated from nodes with the same indentation</td>
<td>R</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>One node per row</td>
<td>R</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All siblings within a box</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tag level</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indentation</td>
<td>R</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Column</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shown in separate path</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Path to tag</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shown on one line</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collapsing</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Different ways structure is shown in the evaluated editors. ‘R’ stands for on request, ‘O’ for optional.
<table>
<thead>
<tr>
<th></th>
<th>Eclipse source</th>
<th>Eclipse outline</th>
<th>Eclipse design</th>
<th>Microsoft Notepad</th>
<th>Outline</th>
<th>Oxygen XML text</th>
<th>Oxygen XML grid</th>
<th>Oxygen XML author</th>
<th>Oxygen XML Tree editor</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tags</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML code</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Icon before name</td>
<td></td>
<td>X X X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inside own rectangle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Between arrow shaped boxes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Name shown</td>
<td>X X X X X X X X X X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Text and CDATA</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML Code</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Icon placeholder</td>
<td></td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Textual placeholder</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inside own rectangle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Between arrow shaped boxes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Content shown</td>
<td>X X X X X X X X X X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Attributes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML code / next to tag</td>
<td>X X</td>
<td></td>
<td>X X X X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Icon placeholder</td>
<td></td>
<td></td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Textual placeholder</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Name and value in different columns</td>
<td>X X</td>
<td></td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name shown</td>
<td>X X X X X X X X X X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value shown</td>
<td>X X X X X X X X X X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML code</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Icon placeholder</td>
<td></td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Textual placeholder</td>
<td></td>
<td></td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Content shown</td>
<td>X X X X X X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Processing instructions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML code</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Icon placeholder</td>
<td></td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Textual placeholder</td>
<td></td>
<td></td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Target in different</td>
<td>X X X X X X X X X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2. Different ways content is shown in the evaluated editors.

<table>
<thead>
<tr>
<th></th>
<th>Eclipse source</th>
<th>Eclipse outline</th>
<th>Eclipse design</th>
<th>Microsoft Notepad</th>
<th>Oxygen XML Outline</th>
<th>Oxygen XML text</th>
<th>Oxygen XML grid</th>
<th>Oxygen XML author</th>
<th>Oxygen XML Tree editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target shown</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Name shown</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Value shown</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

4.3. Possible improvements

Several existing tree visualizations can be used as such or modified for presenting XML files. These visualizations enable new functionalities not present in current editors. For example, alternative visualizations can enable getting an overview of the document and facilitate finding the wanted parts of a large XML file.

Adding tags, attributes, their values and text can be facilitated by using the DTD and schema for limiting what the user can input. Instead of allowing a user to add anything to the document, only content defined as allowed in the DTD or schema could be allowed. This way the document is always kept valid by the program used for editing the document.

In graphical XML editors, text can be shown without using entities, as they are saved in the XML file. Instead of using difficult characters such as “<” and “>” when writing tags, graphical tools can be used for creating and editing tags and other elements in the document.

As XML is text, a text editor is a WYSIWYG editor for XML documents. A graphical editor, on the other hand, is based on an abstraction or visualization of the document. While this can help to getting a better understanding of the document, it can also be confusing for users used to text editors. This has to be taken into consideration when creating and evaluating a new visual editor. A text editor is more suitable for one kind of tasks and a visual editor for other tasks.
5. **Existing tree visualizations and how they can be used with XML**

There has been much research on presenting hierarchical information [Johnson and Shneiderman, 1991; Lamping et al., 1995; Beaudoin et al., 1996; Stasko and Zhang, 2000; Barlow and Neville, 2001]. Most studies concentrate on visualizing directory structures, organizational structures or web page links (the last two are often graphs, not trees). In this chapter, I will discuss existing tree visualization techniques and how they can be utilized when visualizing XML documents. I will focus on finding ideas for building a new visualization combining good features from the existing visualizations.

I will not evaluate three dimensional visualizations, such as Cone tree [Robertson et al., 1991], Fsviz [Carriere and Kazman, 1995] and H3 [Munzner 1997], because they are complex to use and implement. Additionally, current implementations mostly deal with visualizing structures and not editing them. H3, for example, has been able to visualize hierarchies of up to 20,000 nodes on a sphere [Munzner 1997], which could make it interesting when navigating large hierarchies. However, with too many nodes partly overlapping each other on the screen, there is a risk that users get lost. Showing XML data in a 3D visualization is also a challenge. Many 3D visualizations only show a shape presenting nodes, ignoring the name of the node, which is essential when navigating XML documents.

5.1. **Text**

Many XML editors use text for showing the structure together with the content. As the whole file is shown, both content and structure are present. Often indentation and syntax highlighting is used to enhance the view, like in Oxygen XML Editor [Oxygen, 2008]. It also uses a breadcrumb trail to present the path to the current node in some views and this is especially useful if the structure is otherwise hidden, as it is in the author mode.

In addition to indentation there are several possibilities to visualize structures. Many text documents, such as this thesis, are structured with numbers (1, 1.1, 2) or characters (A, B C) to show the structure. One can also use parentheses to present structure. This is shown in Figure 10. [Knuth, 1968, pp. 305-313]

\[(Tag1, Tag2(Tag2.1), Tag3)\]

**Figure 10. Tree structure shown by parentheses.**

5.2. **Node-link diagrams**

Probably the best known tree visualization technique is the node-link diagram. As noticed in Chapter 3, node-link diagrams are already used in some XML editors, and
Microsoft XML Notepad 2007 [Xmlnotepad, 2007] uses it as its only visualization for the hierarchy. By collapsing parts of the hierarchy it is possible to show select parts of large hierarchies. I have discussed these visualizations in Chapter 4.

The problem with the well-known node-link tree is that it does not use space efficiently. It leaves empty space between nodes and even moderate trees require much space to be presented completely [Plaisant et al., 2002]. It, however, works well for displaying small trees and this is why it is so widely used. Plaisant et al. [2002] present an improved node-link diagram, SpaceTree. It allows hiding parts of the tree into triangles. The height of the triangle shows the height of the subtree, the width of the triangle shows the average width of the subtree and the shading the number of nodes in the hidden subtree. Figure 11 shows an example of Spacetree.

![SpaceTree](http://www.cs.umd.edu/hcil/spacetree/orgchart4.gif)


In order to present more information in a window one can use fisheye views [Furnas, 1986] to display parts of interest in detail, and the close surrounding context bigger than the rest of the tree. Lamping et al. [1995] developed the Hyperbolic Browser, a program for navigating and manipulating large tree structures. Nodes are presented as a node-link diagram on a hyperbolic plane with the selected node in the middle. Selecting a node moves it to the center in an animated transition. This way the change of the visualization is understandable to the user. The further away from the center, the smaller the nodes are. This allows showing details of the selected nodes and nodes close to it, and information about the context. Nodes far away from the selected node are small if shown at all. Figure 12 shows how space is distributed to nodes in the Hyperbolic Browser.
While the Hyperbolic Browser works for trees of up to 1000 nodes [Lamping et al., 1995], an alternation is needed for larger trees. With large trees users can get lost and find it hard to navigate in the Hyperbolic Browser. To solve this problem, the root node has been colored in red and always kept visible in Star Tree, an application that visualizes tree structure using the Hyperbolic Browser [Startree, 2008].

The Hyperbolic Browser uses a lot of space for displaying the connections between nodes instead of focusing on displaying information within the node. Fisheye views (focus+context) are useful when dealing with large amounts of data. For example, Jakobsen and Hornbæk [2006] use it for navigating source code and find that users navigate faster with fisheye views and also prefer it over a linear interface.

Node-link diagrams work well for small trees but with some abstractions, as in SpaceTree, large trees can be presented and edited. An improved version of the Hyperbolic Browser could work for navigating XML files. By limiting the maximum length of links connecting nodes, the visualization can be improved to utilize space more efficiently. For visualizing nodes their contents could be presented inside the nodes, allowing the user directly to edit text, attributes and comments within nodes. To enable editing a node that has much information in it, nodes should be possible to zoom
into. This happens automatically on the hyperbolic plane, but for nodes with much content it should be possible to zoom into different levels. Here the idea of Summary Thumbnails [Lam and Baudisch, 2005] could be useful for this zooming. Instead of zooming text, the text is cropped to fit inside an area. This way text is always readable and remains useful. Cropping has to be clearly presented, for example, by using an ellipsis at the end of the text.

Changing a node’s size separately from other nodes could also work if one wants to edit several nodes far away from each other simultaneously. This requires a feature to lock the size of a node so that navigating away from it does not change its size.

I have not found any visualizations dealing directly with navigating in wide trees using the node-link diagram. This is surprising and I consider this a topic for further research. I believe the width is the factor that makes XML files large and it is clearly present in other hierarchies also. Compound fisheye views [Abello et al., 2004] provide one way for clustering and expanding clusters in graphs and this is one solution for displaying wide trees. Instead of showing the whole structure, some parts are clustered and can be expanded in order to navigate in the graph.

Navigating high trees has been solved by abstracting nodes lower down in the tree in SpaceTree, but I believe this technique can also be used for abstracting the parents of a node. An alternative solution is to draw the full tree in a minimized version or simply leave the top of the tree outside the window.

5.3. Cheops, radial presentations and Icicle plot

Cheops [Beaudoin et al., 1996] uses triangles to present layers in a tree. These triangles can be drawn closely together forming a pyramid, where the root is on the top and children are below their parents. This presentation is very space-efficient as nodes overlap, but this makes it impossible in some cases to select a specific leaf, as it might be hidden behind other nodes. Figure 13 shows an expanded tree and its visualization in Cheops. To access leaf nodes, one has to select the parent first if the children are hidden underneath other nodes.

Cheops seems to me as a good way to get an overview of the whole structure but overlapping nodes become a problem when navigating the hierarchy. If used together with expanding a selected level, this could probably be a good way for navigating down and up the tree. For navigating sideways, other methods are probably better.
An effective way for displaying an overview of a tree-structure is to use radial presentation, such as Sunburst [Stasko and Zhang, 2000] or tree ring [Barlow and Neville, 2001]. In these systems the root node is in the middle and generations of child nodes are arranged in circles around it dividing a ring among siblings. Each ring represents one level in the tree. These systems have, however, one troublesome shortcoming. The more siblings a node has, the less space is available for displaying the node and its children [Stasko and Zhang, 2000]. This is apparent in Figure 14 where Sunburst is presented.

Icicle plot [Barlow and Neville, 2001] displays the tree as an icicle where the parent’s width is used to display its children beneath the parent. Like in radial presentations this visualization shows the whole structure well but has the limitation that
nodes deep in the hierarchy are drawn using little space. This can be seen in Figure 15 where the last level has little space for showing the node, even if there is much free space around it.

![Figure 15. Icicle plot with node names shown for the first levels.](image1)

I believe that both tree ring and Icicle plot can be improved by maximizing a selected level to a new visualization. This allows browsing higher and wider trees. Information Slices [Andrews and Heidegger, 1998], a system similar to tree ring but using only half a circle, allows choosing one node as the root node for the next level. This is shown in Figure 16. The program only allows enlarging one subtree at a time but I see no reason for not allowing several enlargements, for example, on top of each other.

![Figure 16. Information slices with one node enlarged [Andrews and Heidegger, 1998, p. 12].](image2)
Sunburst offers three ways for expanding a subtree. The Angular Detail method, shown in Figure 17(a), opens a slice outside the whole hierarchy with the selected node in the center of the slice. Detail Outside expands the slice of Angular Detail to utilize the whole space around the hierarchy. This way the selected node becomes a ring around the hierarchy and shows its children around this ring, as shown in Figure 17(b). The third presentation is Detail Inside where the selected node is shown inside the structure with the selected node in the middle and its children around it. Figure 17(c) demonstrates this.

Sunburst functions well for presenting the whole structure of large trees. It is also functional for navigating high trees with few nodes on each level. However, if there are too many siblings in one level, all nodes are shown as small slices. This hinders navigating between these siblings.

Figure 17. Sunburst’s three ways for expanding a subtree: Angular Detail, Detail Outside and Detail Inside [Stasko and Zhang, 2000, p 81-83].

5.3.1. Suggested improvement: Focus+context Icicle plot

Icicle plots provide a good overview of the whole tree but suffer even more than radial presentations from space limitations. In radial presentations the width of a segment grows the further away from the center it spans when it stays the same in a rectangle. This is demonstrated in Figure 18. The segment 1.1 in Figure 18(a) grows in height from left to right the further the segment spans to right. The rectangle 1.1 in Figure 18(b), however, has the same width no matter how high the rectangle is. Hence, the further down in the tree in Icicle plot, the less space is available for drawing the node.
Figure 18. In a radial presentation a sector grows in both height and width the further away from the center it spans when a rectangle only grows in one dimension.

Oxygen XML Editor’s grid view (that works as nested tables) solves the problem by allowing the visualization to extend outside the window. This way the parent node grows enough to fit the subtrees within the table and can span outside the screen. Instead of this, I suggest enlarging the next level from a selected tag as demonstrated in Figure 19. The children of Node1.1 are shown using the whole available width, as shown by the lines connecting the parent to its subtree. This can hide other nodes, as is done with the children of Node1.2. Node2 is not selected and thus its children are drawn using only the width of the parent.

As nodes are presented as rectangles, fitting conventional rectangular fields inside the nodes for editing its content is possible without empty space around the field. For example, a rectangular text field could allow directly editing names of nodes.

When navigating up the tree, parent nodes could be presented using less space providing a focus+context view. This makes nodes far away from the selected node small and nodes close to it large. In this presentation locking a node’s size could also be a useful feature in order to enable editing nodes far away from each other side by side. In Figure 19 Node 1.1 and its immediate children are drawn bigger than nodes further away from it.

Figure 19. Suggested improvement for Icicle plot with the children of Node1.1 shown using the whole available space.

With the suggested improvements I believe Icicle plots can work well for navigating and editing XML files. However, I have chosen not to evaluate this improved version of Icicle plots in this thesis and left it for further research.
5.3.2. Suggested improvement: Cheops from the upper left corner and combining visualizations

Due to the size of ordinary screens Cheops could start in a corner, for example, the upper left corner. This way there is no empty screen space beside the top of the pyramid. This does not solve the problem of triangular nodes, but if a leaf or selected node is presented as a rectangle, it can start from the tree as the selected node’s siblings also do. Focus+context could be applied to Cheops by making nodes smaller the further away they are from the selected node.

![Diagram of a pyramid with nodes N1, N2, and N3, with N2 enlarged, showing children N2.1 and N2.2.](image)

Figure 20. Suggested improvement for Cheops with the node N2 enlarged.

By enlarging the selected node the rest of the screen can be used in a rectangle for displaying the content of the node. For example, this can be a Treemap of the node and its children. Alternatively the subtree from the selected node can be shown as an Icicle plot using the free space around the tree below or to the right of the rectangle. Showing children above or to the left of the enlarged node breaks the progression of advancing from the top left corner towards the bottom right corner and makes the visualization confusing.

Utilizing space outside the rectangle requires selecting the position of the enlarged node’s children depending on the enlarged node. If the node is on the edge of the pyramid, or close to it, this edge cannot be used for showing children. Figure 20 presents this visualization with node N2 enlarged. Its children are drawn beneath it but the free space above and beside it could also be utilized. If N1 was enlarged, it would not have empty space above it but only under or beside it and N3 similarly would only have empty space above or to the right of it.

Combining several visualizations, such as Cheops, Icicle plot, Treemaps and text can be confusing for a user who does not know the different visualizations well. On the other hand, it also enables utilizing the most suitable visualization depending on the subtree. While Treemaps work well with large trees [Fekete and Plaisant, 2002], Icicle plots show the top-down structure better. XML files have varying structures and utilizing different visualizations for different levels could facilitate handling the document. In this thesis, however, I have decided not to study this further but left it for future research.
5.4. **Nested circles**

Boardman [2000] presents Bubble trees, a visualization similar to Treemaps [Johnson and Shneiderman, 1991], discussed in Chapter 6. Instead of rectangles, nested circles are used to show the structure. When a circle is drawn inside another, the inner circle is a child of the outer node. In Bubble trees it is possible to select a node and make it utilize more space, shrinking the other nodes as shown in Figure 21. This allows browsing higher in the tree and focusing on some part of it while presenting it in context.

![Figure 21. Navigating in Bubble trees [Boardman, 2000, p. 316].](image)

Wang *et al.* [2006] present a circle packing algorithm which resembles Bubble trees and also has a 3D visualization. In addition to showing nodes as circles inside each other, nodes can be drawn as pillars on top of larger pillars. A pillar on top of another one is the child of the underlying one. This visualization has been demonstrated with folder structures, where names of folders or files are shown inside the nodes. Color is used to differentiate between folders and file types.

While circles allow packing nodes effectively inside each other, some space is still lost. In conventional displays, rectangles are more effective. However, the ideas from these visualizations, such as navigation in Bubble trees by maximizing selected nodes, can also work in Treemaps.

5.5. **Summary of the evaluations of tree visualization**

The ways for showing structure in tree visualizations are summarized in Table 3 in addition to showing the findings from Chapter 6. The evaluated tree visualizations have their strengths and weaknesses when answering the questions defined in Section 3.1. For example, while the subtree size and the path to a node are shown well in Sunburst, it is hard to solve these properties in the Hyperbolic browser and Beamtrees.
When comparing the findings in Table 1 and Table 3 it is apparent that more techniques for showing the structure could be used in XML editors. However, different visualizations work for different tasks. Especially showing the subtree size, the number of siblings and the tag level can be improved in current editors. Treemaps are different from the other visualizations in answering all questions and have been chosen for closer inspection.

As the visualizations are for general trees, they only use one way for showing nodes, thus ideas for showing XML content are few. Nodes are shown using some shape in the visualizations and differentiated from each other using color and text.

<table>
<thead>
<tr>
<th></th>
<th>SpaceTree</th>
<th>Star Tree</th>
<th>Cheops</th>
<th>Sunburst</th>
<th>Icycle plot</th>
<th>Bubble trees</th>
<th>Treemaps</th>
<th>Beamtrees</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parent or child</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Textual notation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Lines connecting</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Child inside</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Separate path</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Top connected to parent</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parent inside</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Child overlaps parent</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Root node</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lines entering/exiting</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Whole tree inside root</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colored</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>In the center</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Leaf nodes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Textual notation</td>
<td>(&lt;tag /&gt;)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Missing expansion sign</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Differentiated icon for other elements than tags</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Outmost</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Only nodes shown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Topmost</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Subtree size</td>
<td>Space Tree</td>
<td>Star Tree</td>
<td>Cheops</td>
<td>Sunburst</td>
<td>Icicle plot</td>
<td>Bubble trees</td>
<td>Treemaps</td>
<td>Beamtrees</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>-----------</td>
<td>--------</td>
<td>----------</td>
<td>-------------</td>
<td>--------------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>Spanning shown as border</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One element per row/column</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Numerical notation</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Element size</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Element shape</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shading</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Whole top-down structure shown</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subtree shown in separate visualization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of siblings</th>
<th>Space Tree</th>
<th>Star Tree</th>
<th>Cheops</th>
<th>Sunburst</th>
<th>Icicle plot</th>
<th>Bubble trees</th>
<th>Treemaps</th>
<th>Beamtrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculated from nodes with the same indentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One node per row</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All siblings within a box</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Shading</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Siblings around selected node</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amount of node overlapped by sibling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X X X X X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag level</th>
<th>Space Tree</th>
<th>Star Tree</th>
<th>Cheops</th>
<th>Sunburst</th>
<th>Icicle plot</th>
<th>Bubble trees</th>
<th>Treemaps</th>
<th>Beamtrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Column</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shown in separate path</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Row</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Navigation to root</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Number of ring outside root</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Number of rectangle inside root</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Path to tag</th>
<th>Space Tree</th>
<th>Star Tree</th>
<th>Cheops</th>
<th>Sunburst</th>
<th>Icicle plot</th>
<th>Bubble trees</th>
<th>Treemaps</th>
<th>Beamtrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shown on one line</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X X</td>
</tr>
<tr>
<td>Collapsing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Highlighted via</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thicker lines</td>
<td>SpaceTree</td>
<td>Star Tree</td>
<td>Cheops</td>
<td>Sunburst</td>
<td>Icicle plot</td>
<td>Bubble trees</td>
<td>Treemaps</td>
<td>Beamtrees</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>-----------</td>
<td>--------</td>
<td>----------</td>
<td>-------------</td>
<td>--------------</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>Navigation to root</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rectangles around node</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Ways for showing structure in the evaluated tree visualizations.
6. Treemaps

6.1. Existing Treemaps visualizations

Treemaps [Johnson and Shneiderman, 1991] and its descendants use nested rectangles for displaying hierarchies. Nodes are presented as rectangles and children are positioned inside the parent. Most implementations demonstrate this by showing directory structures (folders and files within folders). Treemaps only show leaf nodes by hiding the structure. In the case of folders and files only files are shown. Nested Treemaps show the whole structure, that is, all rectangles inside each other. Hence, Nested Treemaps show the structure more clearly but consequently use more space for showing the tree. There can be many attributes that determine the size of the rectangle representing the node. The most common solution in the literature is using the file size. Other alternatives are the number of nodes in the subtree, the age of the files and the modification time. Figure 22 shows how Treemaps are used in their first application, TreeViz.

![Figure 22. Treemaps used in TreeViz. Source: http://www.cs.umd.edu/hcil/treemap-history/treeviz.gif.](http://www.cs.umd.edu/hcil/treemap-history/treeviz.gif)

Cushion Treemaps [van Wijk and van de Wetering, 1999], shown in Figure 23, extend Treemaps by utilizing shading inside the rectangle to show the structure instead of nesting rectangles. This saves space compared to Nested Treemaps as no space is used for drawing space between borders of rectangles. This can further be extended to use shading in frames for Nested Treemaps [Bruls et al., 2000] in order to enhance the visualization of the structure. This is shown in Figure 24. In Cushion Treemaps the structure is visible but shading in the frames show the structure more clearly.
When one node has several children in the next level, rectangles of the children become long and thin or low and wide as can be seen in Figure 23. Cluster Treemap [Wattenberg, 1999] and Squarified Treemaps [Bruls et al., 2000] solve this problem by making leaf nodes close to squares. However, this causes two new problems. Firstly, resizing the view can rearrange nodes in the visualization and thus make the whole
visualization change suddenly. Secondly, nodes close to each other, for example two siblings, can be located far away from each other in the visualization. It is thus not possible to know the order of nodes. With directory structures this might not cause a problem but in the case of XML the order of nodes can be important.

The Pivot Treemap algorithm [Shneiderman and Wattenberg, 2001] keeps nodes ordered and Strip Treemaps [Bederson et al., 2002] improve this further to keep nodes ordered and presented as close to squares as possible without wasting space. Bederson et al. [2002] analyze several different algorithms for dividing space between nodes and keeping nodes organized. Even if trees become ordered and there are rules for how nodes are ordered, it is hard, if not impossible to understand the real tree structure from the outputs of the advanced algorithms. Consider the example of Figure 25. Even if it is apparent that nodes are ordered from the numbering of nodes, can a user understand the structure of the tree or the relation between nodes without the numbers? Additionally, all of the advanced algorithms only deal with Treemaps, not Nested Treemaps. Thus, only the leaf nodes are made square like rectangles that are ordered. For Nested Treemaps these techniques do not work equally well.

![Figure 25. Ordered Treemap using Pivot Treemaps [Bederson et al., 2002, p. 847].](image)

Van Ham and van Wijk [2002] have created a different version of Treemaps. Instead of showing relations by nesting elements, Beamtrees show the relationship with overlapping rectangles. If a rectangle overlaps another one, the overlapping rectangle is a child of the underlying node. This makes it possible to display information in both the
parent and its children but without drawing everything within the same rectangle except for leaf nodes. They are drawn within their parent. The presentation can be 2- or 3-dimensional. Figure 26 presents both visualizations of the same tree. This technique prioritizes showing leaf nodes, without the restriction of the level. Even if a node is deep in the structure, the node can be shown using more space than what is used for showing its parent. However, this also makes the visualization more complex than Treemaps.

Figure 26. Beamtrees 2D and 3D. Source:
http://www.win.tue.nl/~fvham/beamtrees/Images/3DFlat.jpg and

Treemaps have proven capable of displaying large trees. Only 3D visualizations have been able to visualize similarly large hierarchies [Munzner, 1997]. Fekete and Plaisant [2002] have used Treemaps to show up to one million items on a 1600x1200 display. This is possible because the presentation of a node is at its smallest the size of a pixel and the structure is shown by positioning similarly colored nodes in rectangles to form a layer. Figure 27 shows 970 000 nodes in a Treemap.
6.2. Tabbed Treemaps

While creating an XML editor that visualizes XML using Treemaps, a new visualization technique called Tabbed Treemaps has been developed. The creation of the editor and the development of Tabbed Treemaps are more closely examined in Chapter 7. Next, the key differences between Treemaps and Tabbed Treemaps are presented.

**Visualization**

Tabbed Treemaps uses Nested Treemaps to show the structure of a tree. Instead of using rectangles, rectangles with tabs represent nodes. The name of the node is shown within the tab attached to the rectangle.
Figure 28. Tabbed Treemaps.

Figure 28 shows the features of Tabbed Treemaps. Children are laid out within the rectangle of the parent, either dividing the space horizontally or vertically. Nodes 1 and 2 are children of the root node, and nodes 1.1 and 1.2 children of node 1. A simple algorithm for laying out nodes is used in order to make it easy to apprehend the structure from the visualization. Nodes can be resized by dragging their borders. A custom sized node keeps its size as long as possible when resizing its siblings, but can become smaller if another node is enlarged. This is demonstrated in Figure 29. Node1 has been resized and is custom sized, as can be seen from is larger size than nodes 2 and 3 in Figure 29(a). When starting to resize Node3, the automatically sized node in the middle, Node2, shrinks first as seen in Figure 29(b). When Node2 reaches a defined minimum size, Node1 starts to shrink. This is the case in Figure 29(c). It is possible to enlarge Node3 until Node1 reaches the defined minimum size. It is not possible to hide siblings completely by resizing nodes this way and nodes are never drawn on top of each other.

Figure 29. Resizing custom sized nodes in Tabbed Treemaps.

If the space for drawing children is too small (smaller than the defined minimum size), children are shown as lines, like the children of node 1.2 in Figure 28. Regular nodes are surrounded by a margin. Nodes drawn as lines can be shown without this margin if using the margin would hide the line completely. This enables showing that there is something inside small nodes. If, however, all children cannot be shown as lines within the parent, only the first lines are drawn. This shows that there exist content in a node without showing how much content.
Navigation
Navigating in Tabbed Treemaps works both by mouse and keyboard. The user can move in the tree using keys. In short, it is possible to move down in the tree with the up key, up (to the first child) with the down key and sideways between siblings with the left and right keys. However, depending on the layout there can be exceptions to this. The idea is to navigate towards the next node in the direction of the symbol on the key that was pressed.

When children are positioned above each other, and they do not have subtrees in them, navigating downwards moves to the next sibling. If navigating downwards in the last child, selection is moved to the parent (the outer node). When navigating to the right from the last node (no matter if children are positioned top-down or left-right in the parent), selection moves to the parent, as this is the next node in this direction. If nodes are positioned above each other and the user navigates upwards from the last node, the previous sibling is selected instead of the parent.

With Figure 28, the navigation can be demonstrated for clarity with some examples. From the root node, pressing the down key navigates to Node1. From Node1 one can navigate to Node2 with the right key and to Node1.1 with the down key. From Node1.1 it is possible to navigate to the text-node within Node1.1 with the down key, to the next sibling, Node1.2, with the right key and to the parent, Node1, with the up key. From Node1.2 it is possible to navigate to its first child, drawn as a line, with the down key, the parent, Node1, with the right key and the previous child, Node1.1, with both the left key and the up key.

A selected node is drawn with thicker borders. If a node that is drawn as a line is selected, it is expanded and drawn as a normal node if there is enough space for this. This can sometimes cause fewer lines to be drawn next to the selected node. The user can this way navigate and view one node at a time among siblings drawn as lines. This is demonstrated in Figure 30. Node 1 has 20 children, of which only the first 18 are drawn as lines in Figure 30(a). When navigating (with the down key) to the first child, as is done in Figure 30(b), the newly selected node is shown using the minimum defined size for nodes. Now only 16 lines are shown next to the selected node. This is due to a minimum spacing defined between the lines (the spacing is unnecessarily big in this situation for demonstration purposes). The user can now navigate sideways to show other siblings.
Similarly to Bubble trees [Boardman, 2000], a node can be expanded in Tabbed Treemaps. This enables showing one node at a time within the parent rectangle as demonstrated in Figure 30(c). By maximizing the node selected in Figure 30(b), the children of node 1 are shown using a row of tabs with one node selected at a time. Now the node previously drawn as line, or using a defined minimum size, is shown using the whole available space within the parent, with the exception of a margin around the node. The selected node’s siblings are shown as tab leafs next to the tab of the expanded node. If there is not enough space to draw tabs for all siblings, arrows can be used to navigate the row of tabs. This is the case in Figure 30(c), where tabs for nodes 1.1 and 1.2 are shown. One can navigate to the other 18 nodes using the arrow keys or by clicking the presented arrow button.
7. A new XML editor

I have not found any XML editors that use Treemaps. This is surprising, as Treemaps are able to display large hierarchies [Fekete and Plaisant, 2002]. In addition to this, I have in my studies and work noticed a need for an improved XML editor. This motivated me to explore this visualization technique more carefully. The goal with this new editor is to test how well Tabbed Treemaps work as the main visualization of an XML editor.

Treemaps can be used almost as such to show the structure of XML files. As the order of elements within the XML file matters one cannot use any advanced algorithms to make nodes close to squares. One has to be able to apprehend the order of elements from the visualization. Thus the Treemaps visualization where nodes can become thin lines has to be used.

As found in Chapter 4, some XML editors provide a possibility to edit the content without the structure. I believe this could be accomplished in Treemaps if elements with data are shown as leafs. If the structure is of importance Nested Treemaps can be used. This way the visualization stays similar but offers two modes like some other XML editors do. In this thesis, however, I have not looked into this more carefully. Only a visualization based on Nested Treemaps is evaluated.

As there are no existing XML editors based on Treemaps, there are no predefined answers on how to differentiate between elements. In XML editors with a node-link diagram, icons seem the most common way. In XML text editors coloring is often used. However, no colors can be seen as standardized as text editors use varying colors for different elements. In Treemaps the background color is used to differentiate nodes from each other. However, in an editor, color could be used for various other functionalities, such as search, similarity or selection. Thus, the use of colors should be user selectable.

Development of the editor

The development of the editor started from Nested Treemaps. In Treemaps information about the node is either displayed inside the leaf or in a tooltip when hovering the mouse over the node. For XML files, the name of the tag is of importance. Thus, Nested Treemaps has been altered to show tags as tabs instead of rectangles. The name of the tag is shown in a rectangle clearly separated from the rest of the tag’s content. Tooltips are also used in the editor, but not as the only way for showing information. This change enables showing the name of the tag clearly separated from its content.

Instead of tabs, text within a rectangle on top of the line, as in Figure 31(b) could also be used for showing the name of a tag. When comparing the rectangles for the nodes horizontally Figure 31(a) and Figure 31(b) use equally much space per node, as
the saved space in Figure 31(b) is only outside the outmost rectangle. Figure 31(c), however, saves 50% of the tab height per level compared to the other visualizations. As seen in the figure, this makes the visualization use more space sideways instead. In order to keep the visualization similar when nodes are maximized and because the denser visualization uses more space sideways, the visualization in Figure 31(a) is chosen for the new editor. However, especially with high trees it can be necessary to use a more compact presentation.

![Figures 31(a), (b), and (c)](image)

**Figure 31. Three alternatives for placing the name for nodes.**

Attributes are shown in an expandable area under the tab. They are not made editable nor is it possible to resize the area. Another possibility could have been to show attributes like tags, but as attributes extend the definition of tags, the selected way seems better. This technique also makes it possible to hide attributes if they are not of importance.

Text and CDATA sections are shown as text within rectangles, similarly to tags but without tabs. Text elements do not necessarily have to show their entire content, instead as much content as fits into the rectangle is shown. When selecting a text element, its full content is shown using a scrollable area. In difference to text XML editors, text is shown without the need to use entities in the new editor. For example, a “<” sign can be written and presented as such, and not using its entity “&lt;” as in text editors.

Processing instructions and comments are ignored and not shown at all. This decision has been made only to save time when implementing the editor. However, the plan is to show comments as grayed out visualizations of the rest of the structure, if possible. Thus, if a part of the structure is commented out, nodes within it are shown in a similar manner to if they were not commented. The only difference could be color – comments would be grayed out. If comments contain an invalid structure, it can only be shown as text. Processing instructions could be presented as empty tags with their contents shown like attributes to tags.
Figure 32. An overview of the new editor. Tags 10, 11 and 12 are enlarged.

Figure 32 shows the XML test document in Figure 1 opened in the new editor. Tags 10, 11 and 12 have been enlarged by dragging the borders of the corresponding rectangles. As can be seen in Figure 32, children are drawn one after another from left to right or top to bottom depending on where there is more space available in the parent. In Tag10 children are laid out vertically, whereas the children in the root tag are vertically placed. This enables concluding the exact structure of the XML document. Alternatively, elements could always be laid out vertically, like in text XML editors, but this would not make use of the screen space equally efficiently.

It could be possible to enhance the presentation by using coloring either in the borders or the background of elements. This has not been implemented, as no standard way for using color has been found. Icons could be used in front of the names of tags to distinguish between different element types, for example, if processing instructions were shown like tags. Thus, with the addition of showing processing instructions this could become necessary.

To enable editing parts of a large XML file, nodes can be enlarged. Bubble trees [Boardman, 2000] make the selected node bigger, while still showing the siblings. In the created editor there are two ways for enlarging a node. Firstly, a node can be resized by dragging its borders, as has been done for tags 10, 11 and 12 in Figure 32. Secondly, a node can be maximized to use all available space inside its parent, except for a margin
around the tag. Maximizing a tag is done by double-clicking its header. Figure 33 shows how the visualization changes after double-clicking Tag9 in Figure 32.

![Figure 33. Tag9 maximized. The text element is being edited.](image)

When a node is resized by dragging its borders the node gets a custom size that is kept as long as possible as described in Section 6.2. When a tag is maximized all siblings are listed beside the tab of the selected tag. Now text and CDATA sections also have tabs. The content of these tabs is the first line of the text element. This view resembles tabs used in several already existing applications. One can navigate the tabs by clicking on them or by using arrows that are present if all elements cannot be shown within the parent rectangle.

Treemaps usually use size on disk to show the size of a node. For XML files alternative information can be used. The amount of text in the subtree can determine the size of the node – this resembles ordinary Treemaps but the number of nodes in the subtree could also be used. However, one does not have to limit to only using this characteristic. The user could determine what tags are important and these would occupy more space. For example, the type of node or user selection could also determine the size in the visualization. This would allow color to be used for showing something else, such as distinguishing elements, but would require a sophisticated algorithm for drawing the nodes. In the current implementation the size of rectangles is equally distributed among the children on each level. This shows the structure clearly, but prevents from using space efficiently. However, after user feedback I expect to change this feature.

If the subtree rooted at a node is too large to be displayed the subtree has to be hidden or abstracted to another form. As found in Section 5.2, SpaceTree [Plaisant et al., 2002] does this by abstracting a subtree into a triangle. In Treemaps, where nodes are rectangles, a rectangle is more suitable. Here color or shading of the node or its borders could be used for showing hidden subtrees. For example, darker nodes could
demonstrate that there is a large tree hidden. This resembles the way SpaceTree presents abstract visualizations of hidden nodes. In the implemented editor heights of hidden subtrees are not shown at all. If a node is too small to be shown, it is shown as a line as described in Section 6.2.

If a tree is high or wide, one has to decide what to show. Is it more important to show the first levels of a tree or only a few nodes but with their complete subtrees within them? The implemented editor prioritizes showing the first levels and the beginning of tag names. Thus, tags could shrink only to show the name of the tag or a part of it, hiding its attributes and textual content. If the available space for showing the tag is too small, it could shrink into a line or a dot. In the implemented editor, lines are used to show wide trees in the editor. If there is not enough space to show all children, as many lines as possible are shown within the parent.

If one level of children is drawn as lines, one can still view the contents of nodes, one at a time. This is possible by navigating to it using keys or searching. The selected node is drawn using the defined minimum size.

In addition to automatically hiding tags as lines and attributes on request, several other ways for hiding parts of the document could be implemented. For example, searching could be improved to hiding nodes that are not among the found ones and nodes could be possible to hide based on their type. With deep structures, the interface becomes confusing with too many similar lines. This could be improved, for example, by using shading.

It is possible to navigate in an XML tree by keyboard and mouse. Clicking selects and double-clicking maximizes tags. Selected and maximized nodes are highlighted with a thicker border. Attributes can be shown and hidden by clicking an arrow. After selecting a text element it becomes a text field one can edit. This is shown with a blue border, as in Figure 33. Arrow keys can be used to move the selection in the editor as described in Section 6.2. There are also shortcut keys available to rename, delete and add elements in addition to performing normal file operations.

Tags and text elements can be added from a context sensitive menu. By right-clicking between two tags, it is possible to add a new element there. Any type of element can be added in the implemented editor but this could be changed to be restricted by a DTD or schema. Using this menu it is also possible to rename tags and delete elements.

Searching in the document is done using a search bar on the bottom of the screen. It searches while the user types and selects the first matching node. If a node is not shown on the screen, due to a grandparent of it being drawn as lines, the closest visible grandparent of it is selected instead. There is also a functionality to select all matching nodes.
**Suggested improvements**

While creating the editor, several improvements have been left unimplemented. Next, these are listed as suggestions for further development.

It could be possible to mix several visualizations together and display some nodes as Treemaps and others using another visualization. This should be user controllable to prevent users from getting lost and to keep the editor domain independent. Differentiating between elements could also be done within the same visualization, for example by allowing the user to specify the color, type of lines and font used for showing a specific element or a group of elements.

Text editors allow any changes to the document and do not require the document to be valid at all times. This freedom could partly be implemented in this graphical editor, for example by allowing adding any type of element via a context-sensitive menu. However, because of the way the visualization works, it is not necessarily possible to allow breaking the structure of the XML document in a graphical editor.

In addition to implementing search and replace, more advanced tools could also be implemented for changing several elements at a time. For example, a user could be allowed to select several elements and perform actions on all elements simultaneously.
8. Evaluation of editing XML documents and the new XML editor

The created editor aims at testing new features for XML editors. Thus, it has only a subset of the functionalities of existing editors. For example, comments are not visible at all. Opening an XML file with comments and saving it again loses all existing comments. The limitations are, however, expected not to attract too much attention during evaluation.

8.1. Objectives

I performed an evaluation focusing on getting constructive feedback instead of a usability study. Thus, the results should be regarded as directive and more research is needed to verify their correctness. There were three main objectives in the evaluation.

The first objective was to determine what XML documents users edit and what tasks they perform with the documents. What data do the documents contain and what is done with the data? These findings help to determine what visualization tasks defined in Chapter 3 are essential when editing XML documents.

The second objective was to find out how the current XML editors meet the users’ needs. What editors and what features are used? What are the important features of the editors and what features are possibly missing? Is there a need to enhance visualizing the structure of the documents? This should answer if there is a need to improve current editors and give directions for improvements.

The third and main objective was to get feedback on the created editor. To what degree does the new visualization solve existing problems and improve handling XML documents? How does the implemented editor work with the documents users edit? Does it facilitate handling XML files? How does the new editor relate to existing editors? How can the implemented editor be further improved?

8.2. The evaluation process

Four users who edit XML files in their work were chosen to evaluate the program. They will be referred to by the names A, B, C and D. Users A, B and C worked in the same research group in a university and developed applications that were configured using XML documents. User D worked in the commercial field and processed measuring results from a system. Users A, B and D reported having recently edited XML files daily in their work. User C reported editing XML files only monthly. Users were chosen because of their XML editing experience.

The evaluation took place in users’ work environment, where they usually edit XML files, on their own computers. All discussions during the evaluation sessions were recorded and the new editor collected log files on performed actions.
Firstly, users presented the files they edit and described tasks they perform with them. This was done with the editors the users normally used. Features such as file size, document structure and document purpose were discussed. While discussing the tasks, users also presented how they used their current editors.

Secondly, current editors were examined thoroughly. The different features of the editors were discussed and users explained what features they used for what tasks. At this stage they presented some suggestions for improvement.

Thirdly, the new editor was presented to the users. This was done using the test document presented in Figure 1. Users were guided on how elements are shown, how they are accessible and how they can be altered. Users were also informed that the tested program is in development and can contain bugs.

Fourthly, users evaluated the new editor using their own XML documents. Users were asked to perform the same tasks they had done with other editors and give comments on how the program functioned. The functionalities of the new editor were thoroughly discussed. Users were asked for improvements to the program regarding showing, navigating, searching and editing elements. At this point, users also provided general comments regarding editing XML files.

8.3. Editing process in the test group

8.3.1. Edited documents

Users A, B and C edited the same files or similar files within the same project. These files were 5-100 kilobytes of size (about 100-2000 lines) and contained up to five levels of tags. These files were configuration files for a program. Before using the documents in the program the files were programmatically preprocessed and joined to form a larger document. This document was not used in the evaluation as users reported only editing it using the separate documents.

User C had tried to edit a larger XML document (it is unclear of what size) created in OpenOffice.org using a text editor earlier. This had, however, proven to be very difficult according to the user and no similar documents were available for evaluation.

User D edited files unrelated to those of the other users. User D edited two types of files. Firstly, configuration files for various programs were edited. These files were of similar dimension to those of the other users. Secondly, user D edited XML documents containing measurement data. These documents were considerably larger, 1-17 megabytes (about 20,000 – 34,000 lines). The smaller files contained ten levels describing the measurements. Measurements were located as siblings within one tag. The largest files were similar in their structure to the smaller files, but contained more elements than the smaller files. This caused tags to have more siblings compared to the
smaller files. Thus, the number of levels was the same. This supports the previous assumptions about XML file structure. XML documents grow mostly in width instead of height.

Most XML files were structured using indentation consistently. Only some of the files that users A, B and C edited deviated from this. By writing several tags on one line, similarly as in Figure 34, a complete subtree was condensed to fit on the screen of the users. Users reasoned that this could have been done subconsciously. Instead, the grouping was based on the semantics of the tags. Figure 34 presents a situation similar to what one XML file contained. Tags related to the name are shown on one line and address information on another line instead of showing all tags on their own lines.

```xml
<person>
  <firstname>John</firstname><lastname>Smith</lastname>
  <address1>Address 1</address1><address2>12345</address2>
</person>
```

**Figure 34. XML shown without consistently following the tree structure.**

### 8.3.2. Actions performed with the XML documents

The users who attended the evaluations performed four actions with the XML documents. Surprisingly, all users performed the same actions, regardless of the type of document or task at hand.

*Learn structure.* With a new document, users browsed through the file in order to learn its structure. This was the only situation when collapsing tags was commonly used. Users A, B and C performed this action in Firefox and Internet Explorer only because of the possibility to collapse tags.

*Navigate to tag.* After learning the structure the most common task was to navigate to a tag in order to view or edit its content. In this situation, users already approximately knew the location of the tag and navigated to it either by scrolling or searching through the document. The larger the document, the more common it was to navigate by searching. Users A, B and C used search very seldom. When searching, these users were most interested in finding tags with the same name.

Usually, users searched for tags using their names. Searching for text elements was rare. Only user D reported searching for attributes, even if all users edited documents that contained attributes.

*Navigate to similar tag.* All users commented on having a need to find tags with the same name. After completed viewing or editing one tag, users mainly navigated to another tag with the same name.
Edit subtree. Users edited several elements within the same parent. The logical structure of XML files makes it understandable to deal with one subtree at a time. In text editors, this also became evident from how tags were indented, as discussed earlier and demonstrated in Figure 34.

8.3.3. Used editors and features

Users reported using the following editors: Crimson editor, Emacs, Firefox (visualization only), Internet Explorer (visualization only), Notepad, Notepad++, Microsoft XML Notepad and XCode. Most of the editors were text editors with syntax highlighting. The only alternative visualization to text in these editors was the node-link diagram. Even if DTDs were present in one case, no features utilizing it were used. The only XML-specific feature used was tag collapsing. This feature was present in Firefox, Internet Explorer and Microsoft XML Notepad.

Users B and C used Firefox and Internet Explorer for learning the structure of XML documents only because of the tag collapsing feature. Microsoft XML Notepad was chosen by user D because it shows the XML starting from having all nodes collapsed. All users, who used tag collapsing, reported that it was difficult to use. Users B and C had to collapse several elements in order to hide the unnecessary elements. Most often they only wanted a small subset of the tags to be shown and disliked that they had to perform several actions to reach the wanted situation. Only user D used tag collapsing when editing and viewing documents after learning their structure.

The most commonly used features were copy-pasting and finding. All users reported using copy-pasting often when creating new content. After duplicating an existing subtree, often as a new sibling of the copied content, users edited its content immediately. Finding was mostly used when the users were familiar with the structure and only wanted to return to a specific point in the document to change something. Find and replace was mainly used to fix misspellings that had been replicated because of copy-pasting.

All users reported using commenting to disable some part of the document. User C reported storing different versions of component definitions in the document using commented subtrees. When the component was needed, the comments around the component definition were removed. Similarly, old data was stored as comments in the document. Many users reported that using comments for disabling parts of the document felt like misuse. One user, however, reasoned that comments have been used to disable program code for long, and that this idea also suits XML documents. Comment elements were also used to add notes to the XML document, but these notes were not edited and used as much as comments that disabled some content.
Two different approaches were used to disable parts of XML documents. Figure 35 presents these approaches. Figure 35(a) shows how the comment markup is used around a tag. The alternative way, shown in Figure 35(b), uses parts of the tag definition for the comment. This was a technique I had not seen before. I believe these different notations could be used to distinguish two types of comments.

```
<-- <Tag1>Text</Tag1> -->
(a)
```

```
<-- Tag1>Text</Tag1 -->
(b)
```

Figure 35. Two different ways to use comments to disable content.

8.4. Evaluation of the new editor

In this thesis, only aspects that improve the editor’s functionality are discussed. For example, suggestions on how to improve the visual appeal will not be discussed in detail.

8.4.1. Visualization

After using the new editor for viewing files smaller than 100 kilobytes, all users stated that it provided a good overview of the structure. As expected, users found it more important to show the first levels, starting from the root node instead of showing the leaf nodes or the whole top-down structure. Users B and C found it easy to compare siblings with each other as content from all siblings was shown side-by-side. Both users gave very positive comments regarding the editor’s abilities to show the structure. There are two reasons for this. Firstly, the important part of the structure was shown on the screen. Elements of interest were not drawn as lines. Secondly, the content users were interested in was well balanced in the document. Siblings had relatively similar subtrees. This made it possible to see all siblings of interest and much of their content at the same time on the screen. Users liked this feature and found the implemented visualization to show structure better than the other editors they had tried. The view was even more useful when users searched for all matching elements within the compared siblings. As the siblings had near identical structures, highlighting all similarly named tags enabled quickly finding wanted children in all siblings. Users suggested further improving this by coloring matching elements with red instead of thicker black borders.

Figure 36 demonstrates a similar view to what users B and C tested. After performing a search matching all occurrences of “tag2”, it is possible to easily compare the content of tag2 in all five siblings even if the tag2 is not located on the same horizontal line.
Figure 36. Comparison of siblings and their content in the new editor.

Users found that dividing space evenly on each level enhanced showing the structure. All users, however, found it difficult to enlarge tags by dragging the borders surrounding the tags. This was because in documents with more than three levels, users could not find the correct border to resize easily among too many parallel lines. For solving this, users suggested using shading in the borders, shading in the rectangle or a drop shadow outside the border to create an effect of depth that changes for each level. Users reasoned that this would make the tree appear to grow up towards the user or down further away. This way borders would differ from each other and enable finding the correct one. Alternatively the shading could vary on every second level.

The problem with too many lines reoccurred when tags could not be shown inside their parent but were drawn as lines. Users found this confusing and tiring to look at. Showing elements as lines did not give enough information about what was hidden. Instead of lines, users suggested using shading inside the parent to show how much content is hidden. The darker the visualization, the more content is hidden. This would make large parts stand out from the visualization. User D also suggested showing this information as numbers. Tags could have a status bar on the bottom of the tag that could show, for example, the number of elements in the tag’s subtree.

Because of showing nodes as lines, the editor was not suitable for the files of over one megabytes that user D edited. Especially with uneven documents, this became a
major problem as a single line could hide thousands of elements. User D, however, expected that this can be solved by using a visualization that highlights parts with much content.

All users were used to XML progressing, as in text editors, from top down. Thus, they found it confusing when a level progressed from left to right in the new editor. However, they also thought they would get used to this easily.

User C tried resizing the window and found the sudden change from one layout to another frustrating. This was because the new layout had more lines than the first one and the transition was sudden. To correct this problem, the user suggested showing the transition using an animation.

Users had many recommendations on how to improve the visual appearance of the program. Most suggestions were taken from existing applications. For example, the search functionality had a button for selecting all elements matching the query, as shown in Figure 32. Users did not find this button easily as it did not look like a button.

Color is not used much in the implemented editor. Only a selected text node is shown with blue borders when editing its content. During the evaluation several ways for using colors were discussed with the users. User C would have liked to color all tags, using one color for each tag type. This user found it important that the program does the coloring automatically. Users A and B only wanted to color tags with the same name as the currently selected tag whereas user D only wanted to highlight text elements using coloring.

Next, I will discuss how to improve showing different XML elements in the editor.

**Tag elements**

In addition to the discussed improvements about showing structure, users did not have ideas on how to improve showing tags. For example, showing the name of tags inside tabs was functional, according to all users.

**Text elements**

Most users thought that showing tag elements as rectangles with tabs and text elements as rectangles was a good way to differentiate between the two elements. Only user D would have liked to emphasize text elements with color. However, the number of rectangles inside each other was a concern to all users. With more content there were too many similar lines next to each other.

When editing documents with encoded characters the editor was, as expected, found to be better than text editors. User A edited a JavaScript code in the XML document that had been difficult to edit in a text editor. In the new editor all characters were shown without encodings and this made the code easy to read. In addition to this, user A found
it particularly useful that the script was within one box as this separated the JavaScript code from the rest of the document.

Attributes
In general, users found the way for showing attributes good. There were only a few improvement suggestions. Firstly, attributes were written too close to the borders surrounding it on the left side. Secondly, users would have wanted attributes to be shown by default with the possibility to hide them one-by-one and all at once. The possibility to hide attributes was highly appreciated.

With attribute values of more than one word, users found it hard to separate between attribute names and values. Users suggested adding quotation marks around the values or showing only one attribute-value pair per row to solve this problem. Figure 37 shows this problem in the current visualization. Tag1 has two attributes, attribute1 and attribute2. Attribute1 has a value containing two words. In this situation it is hard to know if the text “attribute1value2” is the name of an attribute or a value.

![Tag1]
tag1
attribute1=attribute1value1 attribute1value2 attribute2=attribute2value1

**Figure 37. A problem with showing attributes and their values.**

Comments
Comments were actively used by all users. They mainly used comments for disabling parts of the document either to disable the part completely or to switch between two different versions of content. Commenting on the document was also used, but not to the same extent. Thus, users wanted to see comments as grayed out visualizations.

User B suggested an alternative visualization to showing comments as a grayed out visualization. This visualization was suggested to be used for comments that cannot be visualized because they have invalid structures. These comments could be shown using icons as placeholders, with the size of the comment shown in the size of the icon. The content of the comment would be accessible by hovering the mouse over the icon or double-clicking it.

Processing instructions
No user reported using processing instructions. The declaration was the only processing instruction present in the evaluated documents and no user reported changing it.
8.4.2. Navigation

As found in Section 8.3, users wanted to get an overview and learn the structure of the document when starting to edit it. According to users A, B and C the new editor provided good support for this task also when the users needed to navigate in the structure to view the leaf nodes. For most of the files the whole structure was shown on the screen or at most two tags needed to be enlarged in order to see the leaf nodes. In this case users successfully maximized tags above it in order to see the wanted elements.

Unfortunately, no user used XML files sized between 100 kilobytes and one megabyte, so it is unclear how suitable the editor was for files of this size. With the larger files user D edited, which were over one megabytes of size (over 20 000 lines), navigation issues hindered getting an overview of the structure. This was due to two problems. Firstly, when the user enlarged a tag with several siblings before it, the selected tag was hidden outside the screen as the first siblings in the subtree were shown instead. This caused confusion and made it hard to navigate to the enlarged tag’s close siblings. This was a bug in the program and should be fixed in future versions. When the tag to maximize was within the first siblings, this problem did not occur. Secondly, with hundreds of children within a tag, all children were drawn as lines and it became difficult to navigate to them. Possible solutions to these problems are discussed in Section 8.5.

When navigating to a tag in order to edit or view it, users reported that the interface felt intuitive and resizing nodes by dragging their borders was a feature with much potential. However, users did not use this feature much because they failed to find the correct borders to resize. Instead, users maximized tags by double-clicking them.

After maximizing tags, all users found it difficult to navigate towards the root. Users tried double-clicking parents and grandparents of maximized tags in order to navigate back to them. In contrary to all users’ beliefs, this caused only the double-clicked tags to be contracted. Instead, users expected double-clicking to remove all maximizations in the tag’s subtree. Users reasoned that when double-clicking a tag, they wanted to navigate to it.

User D tried expanding ten levels within each other. This used almost half of the screen space on showing the structure, leaving little space for the content of the wanted tag. This problem did not occur with smaller files. To solve this, the user recommended changing this to show the grandparents as a path on one line instead. This would enable using more space for the one wanted tag deep in the structure.
Keyboard
User A stressed that navigating the structure from the keyboard is important. When the mouse works well with a new program, the same functionalities should be possible from the keyboard. Otherwise it is slow to use the program.

Users found that navigating with keys was surprising at first, but after getting used to it, it worked well. The only suggested improvement was by user C, who expected to be able to change the level with the page up and page down keys instead of the up and down keys.

Search
In the new editor, the possibility to search and select all matching elements was met with enthusiasm. This enabled highlighting tags with the same names and facilitated comparing tags. It also enabled getting an overview of where elements existed in the document. Users had not seen this feature in any other editors. Especially with documents that were small enough so that all elements were shown on the screen, this functioned well. To further improve searching, user A suggested enlarging elements that match the search terms. Non-matching elements should not disappear, but be as small as possible in this case. This would allow showing as many matching elements as possible on the screen.

Because searching worked well, users used it for navigation more often than in text editors. Searching for a specific tag worked well when the tag was shown on the screen or when it was accessible by maximizing one level. This was because when searching for a tag not present on the screen, the first visible elder was highlighted instead. With one level this was not a problem, because after enlarging the parent the tag was shown. Searching for tags deeper in the structure did not work equally well, as the selection was lost once the elder was enlarged. Users suggested changing this by preserving the highlighting of matching nodes even if the user selection changes. User D also suggested navigating to the matching element. As in Microsoft XML editor, searching and pressing the Enter key should enlarge tags enough to show the matching tag in the visualization.

No user had used advanced search functions, such as regular expressions when searching in XML files. Users C and D would have liked to define which elements to search within. User D demonstrated the need for this with trying to find a tag with the name “id”. As the string “id” often appears in other words, the user would have wanted to search either for the string within tag names, or so that this is the whole name of a tag. User C suggested developing this further to search for tags by their names that do not contain a given text. The suggested notation for this was “tagName=!text”; search tags
of the name “tagName” that do not contain the text “text”. The user stated that while this need was rare, it was a time consuming task in current editors.

### 8.4.3. Editing

When editing XML documents in the new editor, users expected to be able to drag and drop tags. Users commented that with this addition the interface would feel even more intuitive than now. It would also make this a direct user interface.

As noted in Subsection 8.4.1, comments were often used to disable and enable parts of the structure. Thus, users suggested making it easy to add and remove comments around elements. According to the users, a new option in the context sensitive menu, which was already present in the editor, would be a suitable place for this.

As found in Subsection 8.3.2 users usually edited one subtree at a time. In the new editor this was also apparent, as users maximized a tag before starting to edit content in its subtree. This made it possible to ignore all other elements and to use as much space as possible when editing the wanted elements.

All users were able to add and remove elements using the context sensitive menu. However, as user A pointed out, when using an editor for a longer time, it is important to be able to perform all actions quickly from the keyboard.

When adding a tag, user D expected to be able to add attributes with values at the same time. The user tried to input the string “tagname attributename='value’” as the name of a tag. As a tag cannot have a space inside its name, this could be allowed. From this string it is apparent that the attribute and its values are separate from the tag name.

Users A and C suggested using the implemented visualization for navigating but editing content like in text editors. Both users suggested using this visualization and a text editor in parallel. Selecting an element in one view simultaneously selects it in the second view. User A also suggested adding a possibility to choose the visualization of the subtree of a tag. In addition to the implemented visualization, a text editor could also be used. This way both views can be present within the same visualization. In this case the text editor would be scrollable in order to enable showing any amount of content in it.

As copy-pasting was used much for creating new content in other editors, this feature was also suggested for this editor. User B suggested enhancing the use of copy-pasting by highlighting the pasted content. The highlighting would then disappear once an element has been selected or edited. This way it would be possible to see what pasted parts still can contain false data. User B also suggested creating a possibility to copy only the structure, that is tags and attribute names. This would make it possible to create new subtrees without false information in them.
8.4.4. Other features
When editing XML files, users edited several XML files in a single editor. Users A, B and C edited several smaller XML files that were combined before use in the target program. User D reported viewing one file and creating other files with the help of the original file. Having all XML files open in their own windows would use much space in the task bar and would hinder using other programs. Thus, user C suggested making it possible to keep several files open at the same time within the editor. User C reasoned that this was how other editors also worked.

There were also many specific needs users had when editing XML files. In this thesis I have only discussed those that can be classified as domain independent. For solving other needs, user C suggested adding a plug-in interface, such as JavaScript or Emacs Lisp, to the program.

8.5. Discussion on the suggested improvements
During the evaluation several improvements were suggested for the editor. Next, these changes are presented and discussed.

8.5.1. Visualization

Replace drawing nodes as lines with a better visualization
With large files, a part of the document is inevitably hidden. The current implementation shows elements as lines when they cannot be shown using Tabbed Treemaps. Based on the evaluation this should be changed, but it is unsure how.

All suggested ways for showing content that cannot be presented the normal way suggest that it should be abstracted to one visualization showing how much content is hidden. This, however, is a situation I want to avoid. What if one node has thousands of nodes inside it? Why should these nodes be shown as one visualization if there is space to show more, for example, a subset of the nodes?

Current XML editors solve the problem by using a scrollable space. Can this technique be used in Treemaps? The whole tree could be shown on a scrollable area, but this would cause parts of the tree to be outside the screen. Thus, the benefit that Treemaps provide an overview would be lost. Getting an overview of the structure in the new editor was highly valued by the users. Instead, scrolling could be used for individual tags in situations where children were drawn as lines in the currently implemented editor. In this case all children could be shown using a minimal presentation, for example, only with their names. This, however, only moves the problem with hidden subtrees to the next level. A representation of one of these children can still hide a large tree in it. Thus, scrolling does not solve this problem.
An alternative to scrolling is to cluster the tags and allow expanding these clusters in order to navigate to them. However, how can these tags be clustered in a way that users understand that the clusters are not tags themselves? This has not been studied further, but left for future research.

Treemaps are able to show nodes in a space efficient manner and thus it seems suitable for solving the presented problem. If space is divided equally on each level, as currently done in the editor, Tabbed Treemaps could be used as long as all elements on a level can be shown this way. When there is not enough space for Tabbed Treemaps, some version of Treemaps, for example Squarified Treemaps, could be used instead of drawing nodes as lines. This allows using a single pixel as the smallest presentation of a node. In order to enable using color for highlighting nodes, only a grayscale palette would be used to distinguish non-highlighted nodes. Figure 38 shows a sketch of how this would alter the visualization of Figure 32.

As can be seen in Figure 38, the visualization of the children of Tag9, Tag11Level7, Tag12.1, Tag12.2 and Tag12.3 has changed. When the size of an element is large enough, the name or content is shown inside the rectangle. This is done when showing the children of Tag9 and Tag12.1. For the other elements, the user has to hover the mouse over the element in order to see information about it.

When showing children of a level, all children are not always necessarily shown. This is the case of the children of Tag12.3 in Figure 38. In this case every nth element is hidden in order to ease navigating to any position within the children. This makes it impossible to navigate to all children of Tag12.3 with one enlargement. For example, if every second element is hidden, tags Tag12.3.1 and Tag12.3.3 can be shown next to each other. To navigate to Tag12.3.2 the user would then have to enlarge one of the shown tags first and then navigate sideways to reach the wanted tag. While the presented technique can solve navigating to the next level, one has to enlarge one level before being able to navigate further down in the tree. Thus, research on this matter continues still. The suggestion has not been implemented and evaluated but left for further research.
Figure 38. A suggested variation of the XML editor where nodes are not shown as lines.

Indicate where there is much content in the document

Users expected large parts to stand out of the visualization. A suggested improvement was to show this using shading in the background color of a tag. This is a suitable way when the tag cannot be shown as Tabbed Treemaps but instead shown as a colored rectangle in a Squarified Treemaps visualization. Alternatively the shading could also be in the frame around the tag and also when showing elements as Tabbed Treemaps. However, this alternative cannot be used if the level of the tag is shown in the frame.

Treemaps visualizations use the size of the rectangle for showing the size of the subtree. This indicates where most of the content is on each level. However, the current implementation, which divides space equally between elements per level, was considered good as it enhanced understanding the structure of the document.

Show the level of a tag in its presentation

Users wanted to see the level of a tag in its presentation. They suggested using shading in the borders, shading in the rectangle or a drop shadow outside the border for this. As a tag can contain content inside it, using the background color of the rectangle can hinder showing the content. However, if the content is hidden, the background color is a suitable presentation. A drop shadow outside the rectangle uses more space than if the
presentation is in the border. Thus, showing this information in the border seems the most suitable way.

**Show more content on the screen, vary size depending on the content**

It could be possible to create an advanced algorithm to divide space between nodes unevenly, making nodes with little content smaller than those with more content. All users did not, however, find this a wanted feature as dividing space evenly between each level makes the structure clear. With the level shown in the border of a tag this could change. Thus, after improving the visualization to show the structure more clearly, this feature can be of more interest.

**Enhance finding the borders of an element to resize**

In order to ease finding the border of an element, users suggested varying the borders of elements. While this can partly solve the problem, highlighting borders when hovering the mouse over them can also solve this problem.

**Remove unnecessary borders around single text elements**

In addition to the already discussed ways for distinguish levels, borders of single text elements within tags could be hidden to simplify the interface. All users found too many similar rectangles to be confusing. Other XML editors, such as Oxygen XML editor’s grid view [Oxygen, 2008] and Eclipse’s design view [Eclipse, 2008], show single text elements and text elements with siblings differently.

**Improve visualization of attributes**

Attributes should be shown by default with the possibility to hide them. They should also be shown with a margin on the left side of them, in order to make the visualization consistent. When showing attributes and values, the value can be confused with a name in some situations, as demonstrated in Figure 37. To solve this, users suggested showing one attribute name-value pair per row or adding quotation marks around the value.

**Show comments**

In the current implementation, comments were not shown at all. As most of the comments users edited were disabled content, users suggested showing comments as grayed out visualizations whenever possible. If this was impossible, due to invalid content, users suggested showing comments using icon placeholders. These placeholders would show the amount of hidden content in the size of the icon. The content of these comments would be accessible by hovering the mouse over the icon or clicking it.

In the evaluation, two different ways for creating comments around tags were found. These are presented in Figure 35. These two ways could be used to differ between two
types of comments. For example, the method used in Figure 35(a) could be used for comments shown as icons and Figure 35(b) for those that are visualized. However, the technique used in Figure 35(b) is only possible for tags and processing instructions. For example, text elements alone cannot be commented using the methods presented in Figure 35(b).

**Change the interface using animations**
Currently, all changes in interface in the implemented editor are direct. No animations are used. For example, when the layout mode changes for the children of a tag, users can get confused by the sudden change. To solve this problem, users suggested showing the change using animations.

**Show several expanded tags inside each other as a path on one line**
As found by one user, maximizing several tags inside each other uses a lot of space on showing the maximized structure. To solve this, the user suggested showing the path of maximized tags on one line. Windows Explorer in Windows Vista has a folder navigation feature that would fit well into the XML editor for answering this problem. Here, folders in the path are shown on one line and they can be changed from a drop-down menu, instead of the need to navigate up in the structure first. This is shown in Figure 39. Folder1 has five folders in it, and they are accessible via the opened menu. Similarly it is possible to navigate to the siblings of Folder1 via the arrow between Local Disk (C:) and Folder1.

![Figure 39. Windows Explorer interface in Windows Vista.](image)

The path of maximized tags to the parent of the wanted tag in the XML editor could be shown on one line, like in Windows Explorer. The siblings could be presented as normal tabs on a second line. This way the siblings are shown, and it is easy to switch between them. During evaluation users navigated often between maximized siblings, but not between the parent of these nodes and its siblings.
Implement different coloring modes
During the evaluation, users suggested various ways for using color. One user emphasized that coloring elements should be as automatic as possible. Otherwise they would not be used. To solve this, several coloring modes could be present in the editor.

*Color tags with the same name as the selected one.* Users often wanted to find similarly named tags and in the evaluation they used the search functionality to accomplish this by searching for all elements matching the tag name. Instead of using the search, users suggested automatically highlighting similarly named tags when one was selected.

*Color all elements of a certain type.* One user wanted to highlight all text elements. This functionality could be implemented by improving the search functionality instead of a separate coloring mode. This would be accomplished by allowing searching for elements by their type.

*Color all tags differently.* One user wanted to color tags depending on their type. This is similar to how Treemaps show nodes, using color to differentiate between them. With few different tags, this is expected to work well. However, with more different tags it is harder to find colors that differ from each other. It can also be difficult to combine this feature with showing the structure using differently shaded colors.

**Color elements matching a search in red**
Instead of selecting nodes that match a search, they should be highlighted with red borders, according to the users. This highlighting should persist even if user selection changes in order to allow browsing through the structure without loosing the highlighting.

8.5.2. Navigation

**Enhance navigating to children drawn as lines**
When elements were drawn as lines, users had to use keys to navigate to them. Navigating using the mouse was not possible. This was criticized by all users who tried it. When improving the visualization of elements that cannot be shown as Tabbed Treemaps, navigation to the elements should also be improved. As discussed in Subsection 8.5.1, this could be achieved by allowing users to click on the visualization to navigate to the elements it represents.

**Ensure that an element is shown after maximizing it**
When maximizing an element with siblings, the current implementation shows the first elements of the maximized siblings. This can cause the maximized element to be hidden to the right of the row of tabs. The maximized element is then accessible by navigating
through the maximized siblings. Instead, navigation to the maximized sibling should be done automatically when the maximization occurs. The transition could also be further improved by animating the transition from one view to another.

**Improve navigating upwards in the tree**
All users had problems navigating upwards towards the root when the view had maximized tags within each other. Users tried double-clicking a parent or grandparent, of a maximized tag reasoning that double-clicking means navigating to the tag. In this situation, the current implementation maximizes or contracts only the double-clicked tag. Instead, users suggested removing all maximizations in the double-clicked tag’s subtree. As all users expected the same functionality, the program should be changed to work this way. Additionally, animation could be used to show how the visualization changes when maximizing or contracting elements. However, it is unclear if this functionality also should be used for navigating towards the leaf nodes. Should double-clicking an element maximize all tags in the path to it? If so, how would it be possible to maximize tags only per level? By maximizing one level at a time, users can edit several elements high up in different subtrees next to each other. It is, however, unclear if this feature is necessary.

**Improve the search functionality**
The search functionality was found to be better in the implemented editor than in other XML editors users had tried. To improve the search further, users suggested making it possible to combine several search definitions in one query and enabling searching within specified elements. This would make it possible to perform unprecedented searches. For example, a possibility to search for tags not containing a specific text was suggested by one user.

**8.5.3. Editing**

**Implement drag and drop functionality**
All users expected to be able to drag and drop elements to move them in the editor. This would have made the editor a direct user interface according to the users. However, even if this feature is present in Microsoft XML Editor, user D, who used this editor, did not report using it. In addition to this, no user reported reorganizing elements in XML documents often, if at all.

**Enable adding and removing comments around elements**
All users used commenting to disable parts of the structure. Thus, a feature similar to what Oxygen XML [Oxygen, 2008] has was wanted by the users. In the Oxygen XML Editor it is possible to surround comments around any element in order to disable it. A
suitable place for this functionality in the implemented editor is in the already existing context-sensitive menu.

**Enable adding a tag with attributes**
Currently, one inputs the name of a tag in a dialogue box when adding a tag in the new editor. One user suggested enabling adding a tag with attributes in this dialogue. The name of a tag cannot contain spaces, according to the XML specification [Bray et al., 1998]. Thus, a space could be used to distinguish the name of a tag and its attributes. This would make it possible to add a tag with attributes just like in text editors. For example, the string “tagname attribute1=’attributevalue’ attribute2=’attributevalue2’” could then add a new tag with two attributes and their values.

**Enable editing content as XML code**
All users were used to editing XML documents in text editors. Thus, they suggested adding this functionality to this editor. Two ways were suggested. Firstly, a separate text view could be used, allowing users to browse the document using the implemented visualization but edit the document in a normal text view. The views could be connected. Selecting an element in one view simultaneously selects it in the second view. Secondly, the text could be integrated to the current visualization. Tags could have two visualization modes, graphical and textual. The graphical view would visualize the subtree like the currently implemented editor does. The text view would be a text editor within the rectangle of the node. This idea calls for more evaluation.

**Add a possibility to copy and paste content**
All users reported using copy-pasting often when creating content. Thus, this feature was requested by all users. One user suggested extending copy-pasting by coloring the pasted content in order to highlight parts before editing them. After editing or selecting the highlighted parts, the color would disappear, element by element. The user reasoned that pasted content is often false and by highlighting non-processed content, users would know what parts still needs verification. Alternatively, copy-pasting could contain only the structure. The user would have wanted to copy tags and attribute names only.

**Improve showing and editing tags with several attributes**
If a tag has several attributes, only the first ones are shown in the current editor. This should be changed to enable showing all attributes, even if the space for attributes is not sufficient enough. This can be solved by using a scrollable space or a separate window for editing attributes and their values.
8.5.4. Other features

Allow opening several files within one editor window
All users had several XML documents open in their XML editors. Thus, one user suggested making it possible to have several files open in the implemented editor at the same time. This user reported that keeping several separate editor windows open would clutter the task bar much.

Implement a plug-in interface
Users suggested making it possible for users to extend the editor themselves, without altering the source code. A suitable way for this is to implement a plug-in interface. One user suggested using JavaScript or Emacs Lisp for this. Before implementing this feature, more research is needed on the different alternatives.

8.6. Summary of the evaluation
Most users edited configuration files, which were smaller than 100 kilobytes (about 2000 lines). Only one user also edited data exports from a system. These files were up to 17 megabytes (about 34 000 lines) in size. Four tasks were identified when editing XML documents. When users edited new files they started by browsing through the document in order to learn the structure. When returning to the same file later they navigated to a tag, approximately knowing its location in the document. Users usually edited a specific subtree in the document at a time. When users moved from one subtree to another they usually browsed through similarly named tags.

The implemented editor was found to give an unprecedented view of the overall structure of XML documents smaller than 100 kilobytes. Users were able to get an overall view of the document structure and the structures of selected tags. The compact visualization made it possible to view more elements on the screen than in other editors. This made it possible to compare elements side by side in a way not possible in other editors. This was further enhanced with the possibility to select several tags using the search functionality. With files over one megabyte the editor did not work equally well due to the chosen visualization when elements could not be shown using Tabbed Treemaps. With files of this size, it was difficult to navigate the structure and find where most of the elements were. No documents of intermediate size were tested in the evaluation.

Most users preferred navigating documents using the implemented editor but would have preferred using a text editor for altering content. Thus, users suggested adding text editing functionalities to the editor either separated from the visualization or integrated into it.
During the evaluation, users gave several improvement suggestions for the editor. These suggestions are listed in Table 4 and ordered by importance. A larger number presents higher importance. Most improvement suggestions have relatively straightforward answers, but the ones that require more research are marked in the table with an “X”. Table 4 also lists the number of users who requested the feature.

<table>
<thead>
<tr>
<th>Suggested improvements</th>
<th>Importance</th>
<th>Requests</th>
<th>Research necessary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a possibility to copy and paste content</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Color elements matching a search in red</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Improve navigating upwards in the tree</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Enable editing content as XML code</td>
<td>3</td>
<td>3</td>
<td>X</td>
</tr>
<tr>
<td>Enhance navigating to children drawn as lines</td>
<td>3</td>
<td>3</td>
<td>X</td>
</tr>
<tr>
<td>Replace drawing nodes as lines with a better visualization</td>
<td>3</td>
<td>4</td>
<td>X</td>
</tr>
<tr>
<td>Show comments</td>
<td>3</td>
<td>4</td>
<td>X</td>
</tr>
<tr>
<td>Indicate where there is much content in the document</td>
<td>3</td>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>Change the interface using animations</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Enable adding and removing comments around elements</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Ensure that an element is shown after maximizing it</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Implement different coloring modes</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Implement drag and drop functionality</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Improve showing and editing tags with several attributes</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Improve the search functionality</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Improve visualization of attributes</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Show several expanded tags inside each other as a path on one line</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Show the level of a tag in its presentation</td>
<td>2</td>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>Allow opening several files within one editor window</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Enable adding a tag with attributes</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Enhance finding the borders of an element to resize</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Implement a plug-in interface</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>Remove unnecessary borders around single text elements</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Show more content on the screen, vary size depending on the content</td>
<td>1</td>
<td>2</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 4. Improvement suggestions for the implemented editor.
9. Conclusion

I have studied visualizing XML documents in order to create a visual XML editor. As part of this, I defined some research questions one answers when creating a new XML visualization. Existing XML editors and tree visualizations were analyzed using these research questions.

Three improvements of existing tree visualization techniques were presented: Focus+context in Icicle plot, maximizing nodes in Cheops and Tabbed Treemaps. I also discussed the possibility to combine several visualizations. A new visual XML editor was created based on Tabbed Treemaps. The other presented visualizations were left for further research.

The new XML editor was evaluated and found to give an unprecedented view of XML documents. It enabled getting an overview of the structure of the document or a selected subtree. Although not using space optimally, the visualization was more effective in using screen space than current editors. I also gave directions on how to compact the visualization further.

When editing documents smaller than 100 kilobytes, users preferred using the implemented editor for navigating and searching the document over other XML editors they had tried. They were, however, unsure about how suitable it was for editing content as they were used to editing XML documents as text instead. No documents between 100 kilobytes and one megabytes (about 2000-20 000 lines) were evaluated. With documents larger than 1 megabyte navigating in the document and finding where most content was did not work well in the new editor. This was because the editor showed nodes as lines when they could not be shown using Tabbed Treemaps.

The largest files in the evaluation contained tree structures that were more wide than high. Files consisted of up to ten levels of elements but instead had over thousands of elements per level. Large XML documents were hard to edit because it became difficult to choose the wanted element from among several siblings. This problem was present in other editors also. Solving this problem can mean a breakthrough in editing large XML documents.

During evaluation several improvements were suggested for the created editor. Three important issues without straightforward answers were found. I have discussed possible solutions for these issues but these should be validated by implementing them in the editor and performing a new evaluation.

Firstly, there is a problem with how to present a wide level that cannot be shown completely due to space constraints. A combination of using Squarified Treemaps and showing only every nth element could solve this problem. This would allow a user to navigate approximately to the correct position in the subtree before navigating sideways.
to the correct element. Alternatively, nodes could be clustered and navigated to by expanding one cluster at a time.

Secondly, the current visualization does not show the height of elements clearly. Shading in the frames or in the rectangle of the element could answer this need. Using shading in the rectangle, however, can hinder showing the content.

Thirdly, users want parts with much content to stand out from the visualization. As this problem occurs when elements are not shown using Tabbed Treemaps, the background color of the element can be used, as the element’s content is not necessarily shown. Alternatively, the size of an element can indicate the size of its subtree.

To summarize: There is a need to visualize XML documents in order to learn and navigate their structure. Tabbed Treemaps have been implemented in a new XML editor that has been found to outperform existing editors in showing the structure of XML documents. This motivates to continue developing the editor, starting from the suggested improvements.
References


