Aspect filtering as a tool to support conceptual exploration and presentation

Daniel Pettersson

November 2000

TRITA-NA-E0079
CID/NADA/KTH

Abstract

The ability to effectively filter information when presenting large amounts of data is of great interest – but how do you do it in practice? This is the main focus of my work, which includes design and implementation of an aspect filtering toolkit in Conzilla. Conzilla is a tool for conceptual exploration and presentation of digital information.

Conzilla is developed in Java and receives structured information according to a data model, presently based on files written in XML. The information is presented as concepts in mind-maps, *concept-maps*, that the user may navigate through.

The result of my work has been to design a dynamic functionality that filters the content of a concept according to an arbitrary choice of aspects. Dynamics is essential since you sometimes want to change filter depending on the context for a chosen concept. I have implemented a functionality in Java, which uses the data model structure found in the XML files to match the aspects.

This thesis is a part of a distributed learning environment that is being developed at the Centre for user-oriented IT-design (CID), at the Royal Institute of Technology (KTH) in Stockholm. It has been supervised by Tekn. Dr. Ambjörn Naeve, who is a mathematician and a senior researcher in interactive learning environments at CID.

Aspektfiltrering som hjälp vid begreppsnavigering och presentation

Sammanfattning

Att effektivt kunna filtrera information är av stort intresse vid presentation av större datamängder – men hur gör man för att uppnå detta i praktiken? Detta är problemet som jag har jobbat med och därtill göra en implementation i begreppsnavigeringsverktyget Conzilla.

Conzilla är skrivet i Java och tar emot strukturerad information, enligt en särskild datamodell, som idag kommer ifrån XML-filer. Sedan presenteras informationen som begrepp i "mind-maps", *begreppskartor*, som användaren kan söka i.

Min lösning blev en dynamisk funktionalitet som filtrerar ett begrepps innehåll på ett godtyckligt val av aspekter. Dynamiken är viktig eftersom man enkelt vill kunna ändra filter beroende på sammanhanget för valt begrepp. Jag har i Java designat en filterfunktionalitet som utnyttjar strukturen i datamodellen, återspeglad i XML-filerna, för matchning mot olika aspekter.

Examensarbetet är utfört vid CID (Centrum för användarorienterad IT-utveckling) på NADA, KTH och ingår som en del av forskningsarbetet med distribuerade interaktiva läromiljöer. Arbetet har handletts av teknologie doktor Ambjörn Naeve, som är matematiker och seniorforskare i interaktiva lärmiljöer vid CID.

Acknowledgements

Curiosity led me to investigate different projects at CID, which I find has an interesting profile with its staff of mixted skills (computer science, art, human perception etc.) in order to create a genuine environment for user-oriented IT-design.

An unscheduled meeting with Ambjörn Naeve caught my interest, listening to his visions about interactive learning. He has supervised my work in a thankful way, letting my ideas evolve freely with him as a resource to discuss design and complications with. I also would like to thank Matthias Palmér, who together with Mikael Nilsson is the original developer of Conzilla. Without his knowledge about the structure in Conzilla and his profession in Java, my job would have been a lot tougher.

Finally I would like to thank CID for letting me have a great working space of my own during the time of this thesis.

Contents

Chapter 1

Introduction

In terms of latest news in learning and teaching, many ideas involve software tools and a lot of people believe computerized learning will have a great impact on education in the future. But when interested in creating a worldwide usable learning tool, with the ambition to gather expertise information from wherever it's at – using the Internet – you easy end up with a lot of information.

Here is a great problem nowadays, how do you find the needles in the haystack? Assume there are loads of information about a subject of interest, but you often simply look for certain bits of it. There are several attempts on solving this dilemma, e.g. search engines on the Internet such as Altavista or Yahoo. But as we all know, they seldom return perfect links.

However, interesting work is being done by the IMS project [7], developing a platform-independent framework for web-based learning material. As worldwide web presentation of information begin to follow these standards we get deeply improved possibilities for sorting or filtering information in a more professional way.

Conzilla is a learning-tool, developed at CID at KTH, with the ambition of being on the edge in computerized learning. The Conzilla project follows IMS standards and makes a good platform for taking care of the needles in the haystack – which is where this thesis begins.

In this thesis I will first mention the problem of filtering information and under what circumstances I worked. Then finally present the results in Conzilla and a discussion about recommendations and future extensions to filtering.

1.1 Background

There are three major areas of research at CID at KTH and one is Interactive Learning Environments. In 1996 the first project in this area was initiated, called the Garden of Knowledge (GoK), founded and supervised by Ambjörn Naeve.

A key philosophy in the research is to modularise teaching and learning, by modularising conceptual content into *knowledge components,* see [4]. These components will be constructed in an internationally standardized form, in particular following the standards written by the IMS-project. The components make it possible to separate the content and the context of a given concept, with other words "what to teach" and "what to learn". This is contrary to for example a traditional course where the connection between content and context is already set.

In 1998 two students got interested in the GoK ideas and developed the first concept browser under the supervision of Ambjörn. Mikael Nilsson and Matthias Palmér named their tool Conzilla and it was developed as a part of their master thesis work in computer science at TDB (Dept. of Scientific Computing) at Uppsala

University. Conzilla really caught their attention and they have continued developing it, Matthias is now a PhD student at CID.

The amount of discussed future extensions for Conzilla are huge. Among the ideas is, originally described by Ambjörn in [4], the possibility for the user to filter the information of a chosen concept on a dynamical set of aspects. This is where my thesis work takes place – creating a dynamic filter functionality for Conzilla.

1.2 Concept browsers – design principles

Conzilla is described as a concept browser, but what is that? A conceptual organization and presentation scheme that supports the conceptual context will be referred to as a *concept browser*, introduced by Ambjörn in [4]. Ambjörn also presents six design principles for this kind of browser:

(i) Separate context from content.

(ii) Describe each context in terms of a concept-map. A concept-map is a kind of mind-map, relating the concepts in the context.

(iii) Assign an appropriate set of components as the content of a concept and/or conceptual relationship.

(iv) Filter the components through different aspects.

(v) Label the components with a standardized data description (= metadata) scheme.

(vi) Transform a content component, which itself is a concept map, into a context by contextualizing it.

As Conzilla is still under development to become a true concept browser, principle (iv) was not implemented yet and caught my interest in completing.

1.3 Conzilla

As Mikael and Matthias describe their focus [3]:

> "We are interested in the organization of knowledge of any kind, and the presentation of this organization."

Regarding the design principles of a concept browser described in 1.2, they found some further requests when designing Conzilla:

- Platform independent, in order to build a worldwide usable tool they chose Java.

- Information should be able to be presented in any form - movies, text, audio, pictures etc. Regarding the principle (i) in 1.2 implied they separated the organization of knowledge from the presentation of it. They chose the presentation to be shown in an external content displayer, presently in the default webbrowser.

- Wanting easy usability and information in a standardized way, they followed the guidelines given by the IMS-project, which provides hints for computerized learning.

- The information for presentation is structured according to a data model, in order to make it easy to reach and organize. Presently the input information is given in XML files, which is a good markup language for structured information

The tool presents the conceptual relationships between a set of concepts in a model that strongly resembles UML class diagrams, see [13] and [14]. The input data is presented as concepts related in surfable *concept-maps*, and allows the user to view the content describing these concepts. A set of connected concept-maps is referred to as a *knowledge patch.* It could represent an area of interest, such as e.g. music instruments.

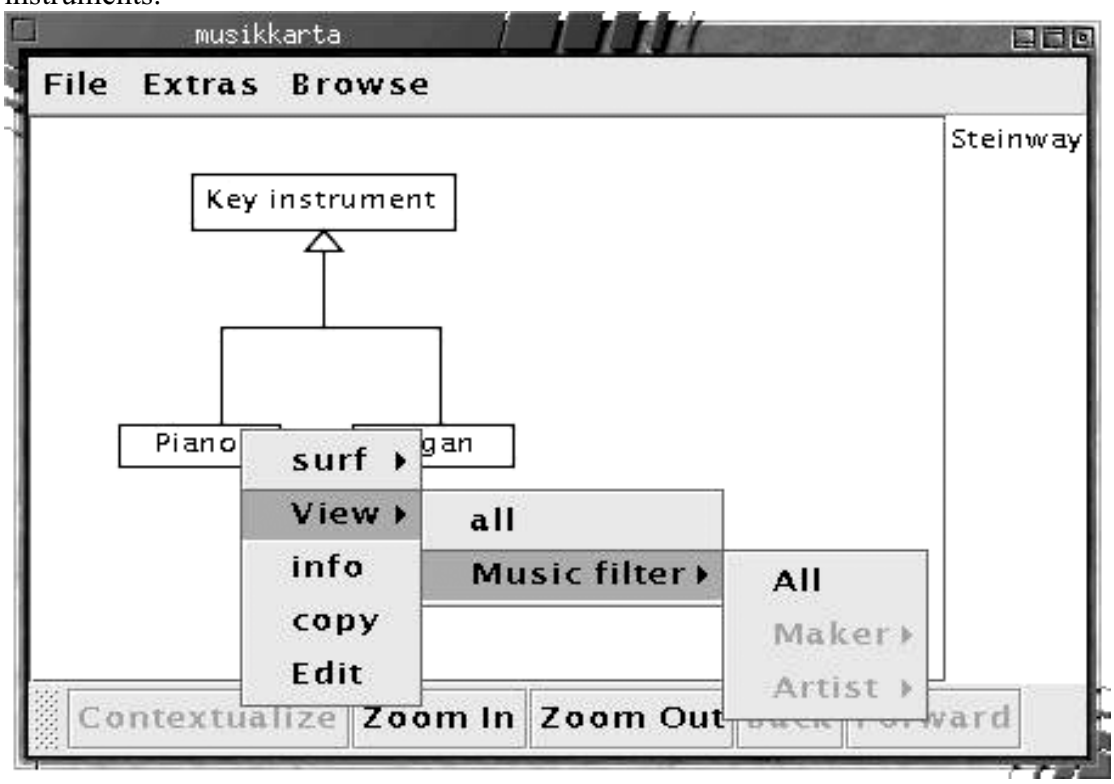


Fig. 1 Conzilla presenting the concept-map "musikkarta".

In order to describe my development in Conzilla I will first present a short guide to the program:

- **Loading a concept map –** First you must load a concept map into Conzilla. Either you give this parameter when starting up Conzilla or you can use the Open map, in the File menu (see Fig. 2).

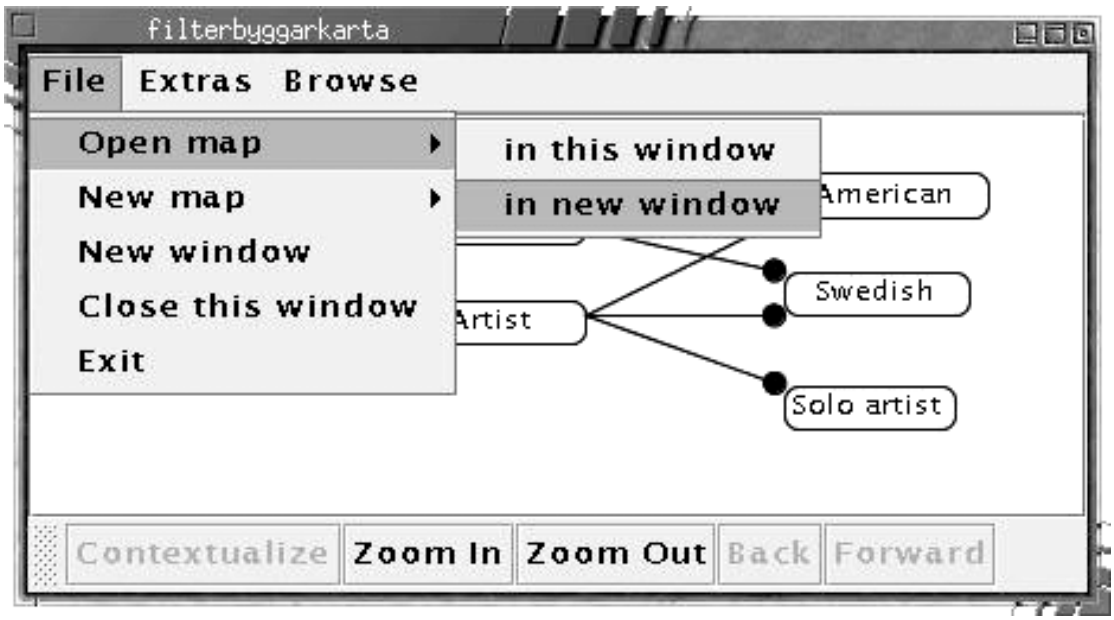


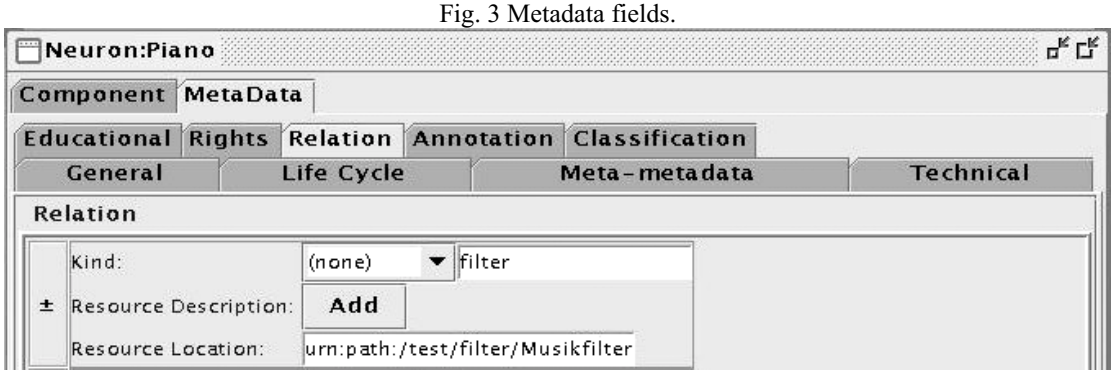
Fig. 2 Open map in Conzilla.

- **Surfing** – To navigate between different concept-maps is called "surfing". You use the mouse to focus on a concept of your interest. Right-click a concept and a menu appears. There you choose the "surf" command (see Fig. 1), which brings up a more detailed concept map for the concept you choose. If the surf command is greyed out, you can't choose it because there is no detailed map available.

- **Viewing the content of a concept** – Right-click a concept and the menu appears. Now choose the "view" command. This will present a list with all different content on Conzilla's right-hand side (see Fig. 1). Pick a content of interest and your default webbrowser will be launched, presenting the chosen content. If the view command is greyed out, you can't choose it because there is no content available for this concept.

- **Displaying metadata of a concept** – Right-click a concept and the menu appears. Now pick the "info" command. This will present a list of metadata about the chosen concept, e.g. saying who the author is and when it was created etc (see Fig. 3).

Fig. 3 Metadata fields.



For further knowledge in operating Conzilla, refer to the homepage[1] of the project. Here you can download the program for free, read the manual, latest news or join a discussion group.

1.4 The filter problem

A serious problem when using Conzilla is that you could get overloaded by information. Suppose you are curious about a concept that is very common and worldwide well documented. Further you are browsing a concept-map where this concept has a large set of content components. As you want to view the content of the current concept, you probably will get lost and bored by the amount of information. Also a common user situation is that you are simply looking for a specific detail or aspect of the concept. For example, you are surfing a concept-map about musical instruments and your concern is to find out all you can about the famous pianomakers Steinway. It is reasonable that a concept-map about musical instruments includes a concept about key instruments and probably even beneath that one, another concept about pianos. Well, there are certainly a lot to know about pianos and if the content of the piano concept is just a bit ambitiously documented, you will spend the rest of the day digging for Steinway related stuff.

This is why an option of filtering the content in some way is of great concern. But what features are relevant for this filter? The idea of filters had been discussed within the Conzilla project for quite some time and after me having understood the problem, together with Ambjörn and Matthias the following features seemed desirable:

(i) Assume that lots of people with specific needs and different knowledge use the system. Hence, the filter functionality must be dynamic. Different concept-maps and concepts must have the possibility of using different filters. Also depending on the current user, the filters must be changeable.

(ii) People have different needs and backgrounds, which calls for a "multidimensional" option of filtering. This needs a closer explanation and is found well described in [4] where the original thought is presented by Ambjörn. Let's assume that if a set of concept-maps represents an educative overview of mathematics and is supposed to be used by all students studying mathematics. The senior researcher and the high-school kid might have the same question about the filter aspect *where* you use algebra, but they would probably have different opinions on at what *depth* the information should be presented. Referred to in [4] as "Multiple scale narration with 2-d resolution based on clarification and depth".

(iii) An easy and understandable way of creating and editing filters within the browser is an attractive and well needed feature.

(iv) Filters should be defined in such way that they are easy reusable and allow for different forms of combination.

[1] http://cid.nada.kth.se/il/conzilla/default.html

(v) They should work without the aid of concept maps, as they are useful in situations dealing only with the database structure, as well as in combinations with other sorts of concept presentations.

(vi) As long as all content is tagged with metadata, the filter definition should not depend on any data format. This allows the filtering of all kinds of digital information such as text, movies, audio etc.

Chapter 2

A possible solution

I had to find an easily understandable way of presenting the filter functionality for a new user of Conzilla. The logical connection for filtering a concept would be with the "view" choice in the concept-menu that appears when right-clicking a concept. If no filter was connected to the concept, a click on "view" would result in a presentation of all content, like before.

So I wanted to find a design that met all filter requests described in 1.4. First find a dynamic way to represent a filter. Then there had to be an understandable GUI. Further there was the question about how and where to connect a filter to a concept. Finally, how to create and edit filters of your own.

2.1 Filter nodes

An initial question was the underlying representation of filters. After some discussions, a good alternative was to represent each filter aspect as a node in a connected tree – a filter tree. Given a connection of two filter nodes only the content that matches the filter aspect of the node A is passed on to B, the rest are thrown away (see Fig. 4).

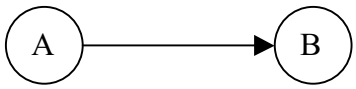


Fig. 4

All the filter aspects that you want to include in your filter are connected to form a filter tree of your choice. The first node just tells the name of the filter and point out the first order of filter aspects. Say you choose some filter aspects for filtering a concept about pianos and create a filter tree of them. Someone picks the aspects *makers* and after that one *american* – then the tree with the chosen aspects, marked grey, could look like Fig. 5. All the content of the concept piano will be filtered through the aspects and only the content components that matched both the chosen aspects will be presented.

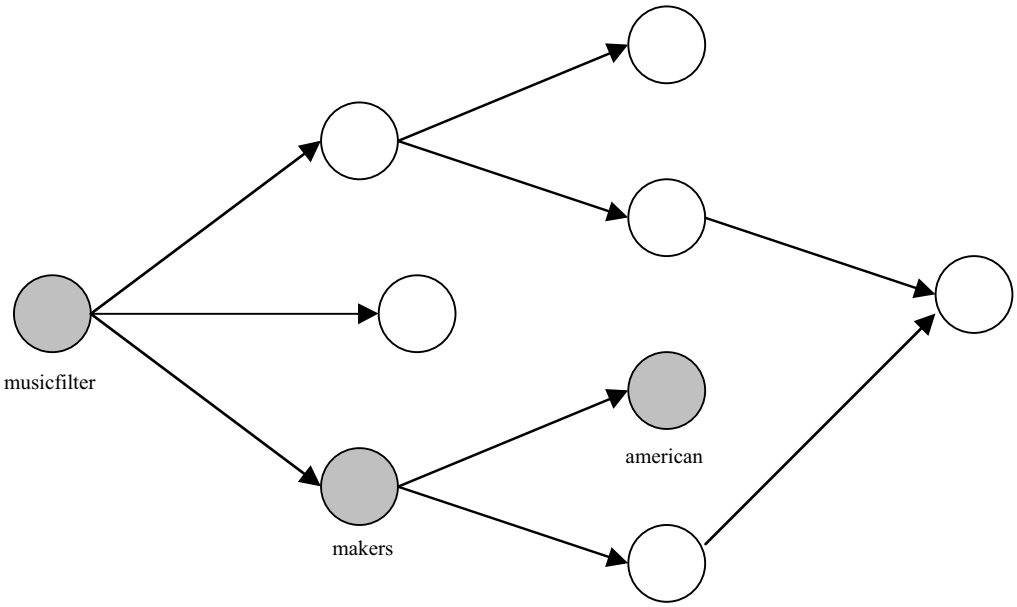


Fig. 5

Representing filter with nodes you could easily connect nodes and reuse copies of old nodes, which match request (iv) in 1.4. Further, several layers of nodes match and extend the multidimensionality request (ii), each layer could represent one dimension. But it extends in the respect that each layer doesn't have to be restricted to the same dimension. For example, if the aspect *history* about a concept has an underlying set of aspects providing different depth of the information, say the aspects *low*, *medium* and *high*, it could also have two other aspect in the same layer saying *modern* and *early* (history).

But there are limitations in this design. Presently, when you use the filter you pick a path in the filter tree and that is the only option. But say you would like to get all the content of the concept piano on both the *history* and the *makers* aspects. Well, presently that is impossible, but one solution is to implement subfilter, further discussed in Future Extensions Chapter 5.

2.2 GUI

What would an appropriate GUI look like when filtering a concept? One major preference was that the GUI had to match request (ii), the option of multidimensional filtering. I had a look at Ambjörns 2-d grid-style presentation for this purpose, found in [4, Fig. 18, p. 35]. This layout would be very pedagogic when it is useful, but rather tricky to implement. Since the programming focus was the underlying filtering algorithms, I decided to go for something easier. A satisfying solution was to recursively use the same sort of menu that the "view" choice appear in. Where as one menu represented one layer of nodes. For example, if you have chosen a concept about piano, the filter could, after the first click on "view", present a new menu with the aspects *makers, history* and *artists.* Say you pick *makers* and beneath that one a new menu appears with the aspects *swedish* and *american* – picking *american* as the final filter aspect could reveal Steinway among others.

2.3 Metadata

There had to be an easy way of changing filter for the current concept or concept-map, 1.4 (i). As each concept and concept-map includes metadata describing the object, here would be a sensible place to put the link for the associated filter. The metadata are easy found in the “edit” choice in the right-click menu, where you also can edit a metadata field, which fitted the filter-link purpose perfectly. So, by giving the metadata field of each concept an option of a filter tag with the URI to the first filter node, a dynamic connection was created.

2.4 Component editor

A final interactive functionality for filters would be how to edit or create new ones. When I began implementing there was no suitable edit-mode provided by Conzilla, resulting in that you had to edit filters in their true file format – XML, using an external editor as Emacs. This was not satisfying, since it forced a non computer professional user to go outside the Conzilla environment and having to face a quite tough job creating filters of his own. Still, presumably only a few Conzilla users will ever create filters – saying people that create a set of concept-maps about an area will most likely be the best creators of filters for that knowledge patch. These people can be expected to have good understanding of Conzilla and have probably heard of Emacs and hopefully of XML. Nevertheless, the Component editor showed up in Conzilla during the final implementation of the filter, providing a nice environment for editing and creating filters within the browser and without forcing the user to interface with XML. Case closed.

Chapter 3

Filters in Conzilla

3.1 The results in Conzilla

The final results in Conzilla were that users have the possibilities to create or edit filters of their own, connect and change filter for any concept or concept-map and finally to filter any concept on a chosen set of aspects.

3.2 Using filters

A criteria for filtering a concept is that there has to be a filter associated with the chosen concept. To begin with Conzilla checks whether there is a filter connected to the chosen concept. If not, it looks for a filter in the current concept-map. The functionality is built to work like this because say in a concept-map about mathematics perhaps the wanted filter aspects look more or less alike for different concepts. Then if there is no filter ever associated in either the concept or the concept-map – you get all content about a chosen concept, as Conzilla used to work before introducing filters.

How do you filter a concept in Conzilla? When you surf an arbitrary concept-map and find an interesting concept, right-click the concept and choose the “view” alternative. If there is a filter associated with the concept, you will get a new menu with the first list of aspects. Pick an interesting aspect and Conzilla will return the content matching your aspect. Sometimes there are multidimensional ways of filtering. If you pick an aspect that lists another menu beside (it will do so if the aspect has a small arrow printed before it), you have the possibility of further refining your filtering choice. Or pick the first alternative “all” in the new list, which returns all the content of the previous aspect.

3.3 Connecting or changing filters

To connect a filter to a concept, you right-click the concept and choose the “edit” alternative in the menu. Then you will get a window presenting all metadata connected to the concept. Choose the “relation” tag and add a relation, see Fig. 3 p. 14. Here you write “Filter” in the “kind” field and the first filter node’s URI in the “Resource Location” field. When naming the URI you can take advantage of the resolver.xml, which could contain shortcuts to the file tree. Resolver.xml can be edited in the Resolver editor.

If you want to change to another filter for any concept, you simply write the new filter’s URI in the “Resource Location” field mentioned above. Further, the same procedures apply for connecting and changing filters to concept-maps.

3.4 Editing filters

Creating a set of filter nodes and connecting them into a filter tree can be done in the Component editor, which is included in Conzilla. To open the Component editor, you pick the Extras menu in the top of the window and there you can choose the Component editor. Here you create each filter node and fill in name, filter URI and the filter tag that is used to filter an object (see Fig. 6). For a better tutorial see [11].

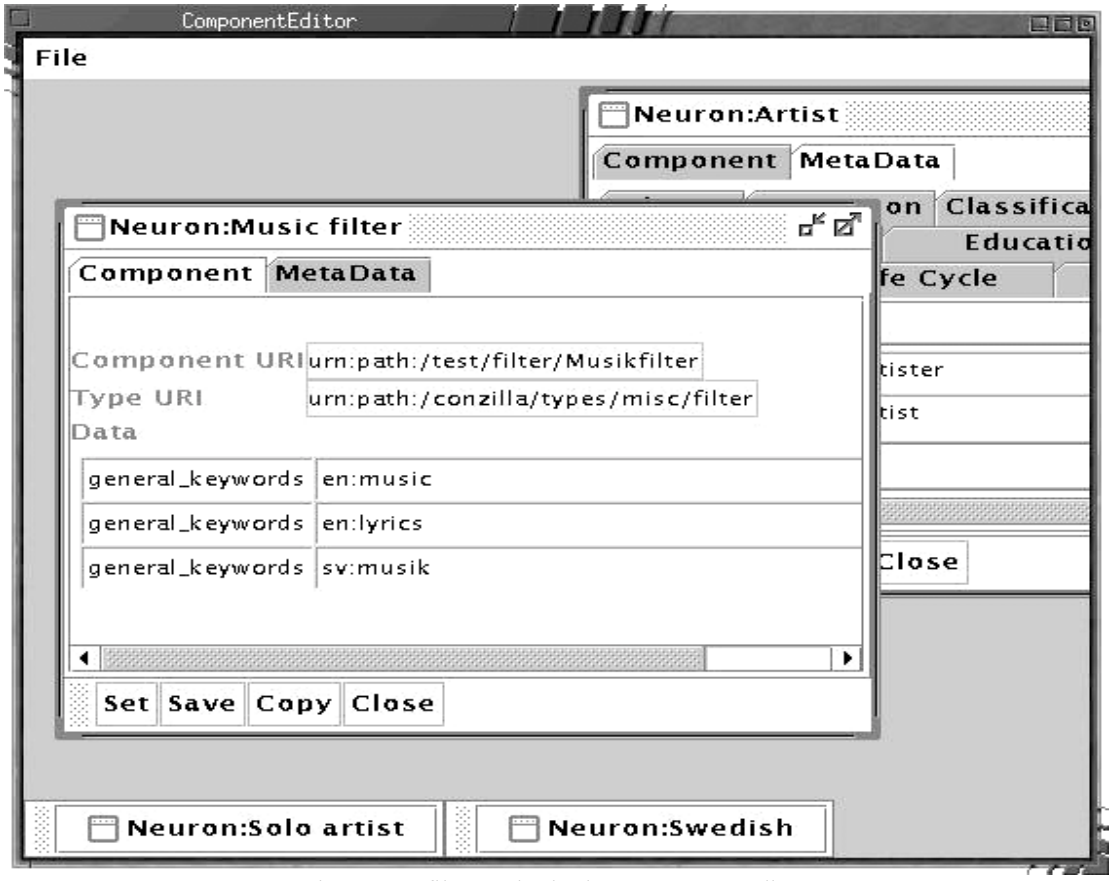


Fig. 6 Two filter nodes in the Component editor.

Chapter 4

Implementation

First I had to find a structure for the filter files that fitted the rest of the browsing material, based on XML and using the neuron thinking developed by Mikael and Matthias. In their model all input to Conzilla are described as neurons related to each other, like in the brain. Then there had to be a functionality within the browser that could read these filter representations, present them and on the user's command be able to filter the content of a concept through the filter.

Conzilla is developed in a standard object modelling fashion with independent modules for different functionalities. So I had to design a structure that fitted the polymorfic pattern, further described in [1] p. 107 Factory method. The design should be as open as possible, in order to easy implement other kinds of filters or even filtering techniques in the future.

All the new java files for the filters were put in a separate directory named Filter.

4.1 Filter neurons

The neurons can include different metadata, data and roles. The metadata describes the specific neuron, the data fields contain the information to use and the role fields describes the neuron's relations to other neurons. In Conzilla there is a type system for neurons. Each neuron is associated with a neurontype. The neurontype for the filter node was named filter.xml and looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE NeuronType PUBLIC "-//CID//DTD NeuronType 1.0//EN"
"NeuronType.dtd">

<NeuronType>

  <MetaData>
    <Tag NAME="Constructor">Daniel Pettersson 000412</Tag>
    <Tag NAME="Language">English</Tag>
    <Tag NAME="Title">Filter</Tag>
    <Tag NAME="Description">The Filter type</Tag>
  </MetaData>

  <DataTags>
    <DataTag NAME="FILTERTAG"/>
    <DataTag NAME="ACCEPT"/>
    <DataTag NAME="REJECT"/>
  </DataTags>
...
```

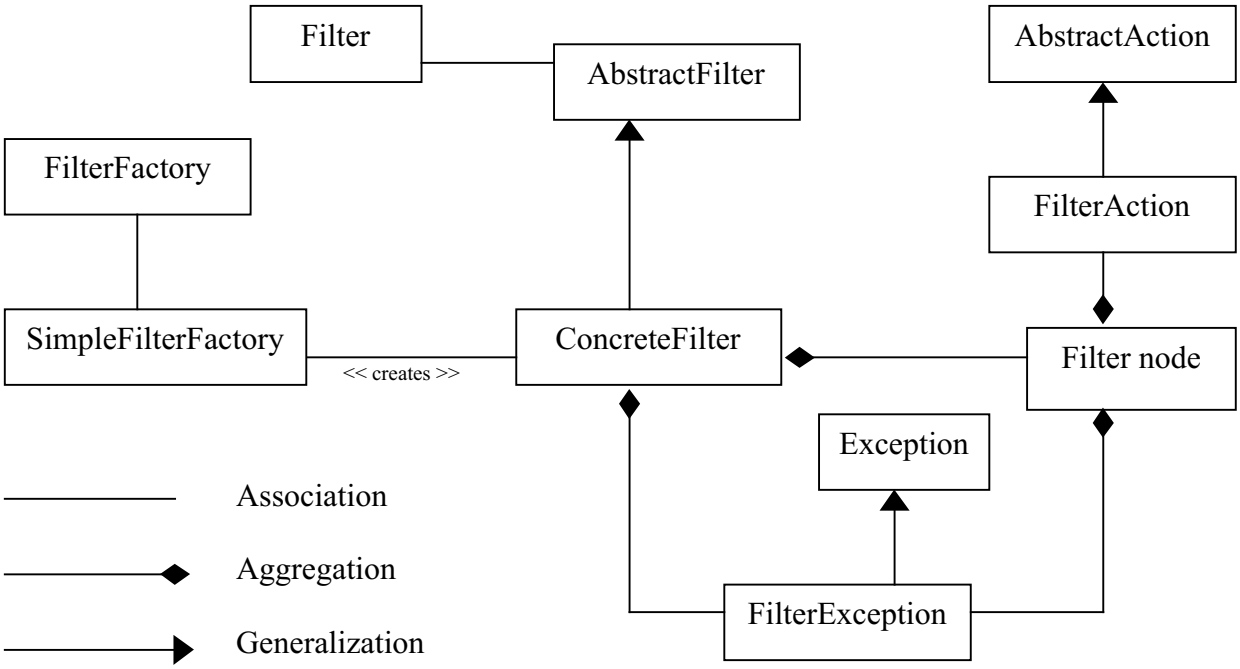
As you can see in the DataTags field it specifies the FILTERTAG. This tag is the one used to filter the content and there is one for each filter node. A typical filter neuron file, here musicfilter1.xml, has the following structure:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Neuron PUBLIC "-//CID//DTD Neuron 1.0//EN" "Neuron.dtd">
<Neuron TYPEURI="cid:local/neurontype/filter">
  <MetaData>
    <Tag NAME="Title">Musikfilter 1</Tag>
  </MetaData>
  <Data>
    <Tag NAME="FILTERTAG">maker</Tag>
  </Data>
  <Role TYPE="refine" NEURONURI="cid:local/ne/musicfilter11">
    <Multiplicity LOWEST="1" HIGHEST="1"/>
  </Role>
</Neuron>
```

Here you see the associated neurontype filter in row three and in the Data field the FILTERTAG is set to "maker". Hence, only content tagged with the word "maker" will pass this filter node.

4.2 UML

Here is a UML diagramm describing the internal relations among the filter files:



AbstractAction and Exception are included in the standard JDK. For further information on UML see [13] and [14].

4.3 API

Class / Method	Purpose

Interface Filter
 	
`String getURI()`
`Filter node getFilter node()`
`void setContent(Neuron)`
`throws ControllerException`
`void showContent(Vector)`
`Vector filterContent(Filter node, Neuron)`
	The interface for the filter class.
	
	Returns the URI string of this filter.
	Returns the first filter node.
	Sets the content for current neuron.
	
	Displays the given content.
	Filters the content of given neuron through given node.

Class AbstractFilter
 	
`public AbstractFilter(MapManager,`
`MapController, String)`
`throws FilterException`
	The abstract filter class.
	
	The constructor for filter.

Class ConcreteFilter
 	
`public ConcreteFilter(MapManager,`
`MapController, String)`
`throws FilterException`
	The filter class.
	
	Creates a filter.

Interface FilterFactory
 	
`public ConcreteFilter createFilter(`
`MapManager, MapController, String)`
`throws FilterException`
	The interface for the filter factory class.
	
	Creates a ConcreteFilter.

Class SimpleFilterFactory
 	
`public ConcreteFilter createFilter(`
`MapManager, MapController, String)`
`throws FilterException`
	This class creates a concrete filter.
	
	Creates a ConcreteFilter.

Class Filter node
 	
`public String getFilterTag()`
`public Filter node getRefine(int)`
`public FilterAction getAction()`
`public int numOfRefines()`
`public Filter node getTop()`
`public void setTop(Filter node)`
	The filter node class.
	
	Returns the filter tag.
	Returns a refined Filter node.
	Returns the FilterAction of the node.
	Returns the number of refined nodes.
	Returns the node above.
	Sets the node above.

Class FilterException `public FilterException(String)`	This class is used to announce an error. Gives the error message to the system.
Class FilterAction `public void setContent(Vector)`	This class updates content. Sets content for current concept.

4.4 Design techniques

- **Factory design pattern –** The demand to easily be able to implement other kinds of filters called for a thoughtful design when creating filter. In [1] you can read a lot about different design patterns used for structured programming. What fitted well here was the Abstract Factory pattern – "Provide an interface for creating families of related or dependent objects without specifying their concrete classes." Thus, as shown in 4.2 I created a FilterFactory interface that supports the making of different filters. Only one kind of filter is being produced so far and that is done by the SimpleFilterFactory, which creates a ConcreteFilter (see 4.2). This also follows the guidelines given in [1] – letting subclasses decide which class to be instantiated.

- **Looped nodes –** When Conzilla reads a new filter of filter neurons, it recursively creates a tree of connected filter nodes. As the filter is represented by connected nodes, there is a risk that a node reappears in the filter tree because it was by mistake connected with a later node. Then the filter is looped and could run forever. To prevent looped nodes to occur I created a loop check. It summons the connected nodes in a LIFO (LastInFirstOut) stack when the filter neurons are being read. If a node is found in the stack before being put there – there is a loop.

- **Recursive filter method –** When a user has chosen a set of aspects to filter the content of a concept, only the content components that match all the aspects is presented. An aspect is matched if any of the keywords of a content component match the filter tag found in the filter node (= aspect). I used a recursive solution for parsing the aspects, see method `recursiveContent` below. First it finds the top of the filter tree, and then each content component is matched with the current aspect. If they don't match the component is thrown away and when the last aspect have been parsed, the remaining content is presented.

```
private Vector recursiveContent(Filter node node)
{
  if (node.getTop() != null)
  {
     contents = recursiveContent(node.getTop());

     for (int i=0; i < contents.size(); i++)
     {
        keywords = (contents.elementAt(i)).getValue("keywords");

       if (keywords == null)
          contents.removeAllElements();
       else if (keywords.indexOf(node.getFilterTag()) == -1)
       {
         contents.removeElementAt(i);
         i = i - 1;
       }
     }
     return contents;
  }
  else
     return contents;
}
```

Chapter 5

Future Extensions

Through the work of this thesis there has been some discussion about complementary details that you could include in the filter functionality. Some were implemented, some weren't good enough and some were simply too extensive for this thesis. Here are some possible future filter build-ups:

- **Complete filtering on metadata -** Not only keywords are relevant when filtering a content component. There are lots of possible metadata to tag on contents and of course these sometimes make good filtering alternatives. For example, the author of a concept, when was it made, what kind of data format is it etc.

- **Subfilter -** There are limitations in the filtering functionality, as mentioned in 2.1. For example, you can only pick one path in a filter tree. If you want all the content that passes two or more different paths you have to filter several times and look at the results separately. A solution to this problem is to introduce subfilters. The idea is, for a content to pass a filter node with a subfilter attached to it, the content first has to pass the subfilter and second the filter tag of the current node. A subfilter looks like an ordinary filter of connected nodes, but because it is related as a subfilter it is used in a different way. To pass a subfilter means passing one of all the possible paths in a subfilter. The example mentioned in 2.1 illustrates this idea. Suppose you would like the concept piano to be filtered through both the *makers* and the *history* aspects. Then one of the first layer nodes, named X, is connected to a subfilter and this subfilter has the two nodes *makers* and *history*. Hence, the content that passes either *makers* or *history* in the subfilter will be presented when choosing the aspect X.

- **A library of filters -** Presently the idea is that the author of a knowledge patch provides some appropriate filters and the user has the possibility to make his own. But nevertheless it would be an advantage with a library of pre-produced filters that could cover some common areas.

- **Filter editor -** The main purpose of the Component editor is to provide a good environment for producing different components. As mentioned, it may be used to produce filters, but for example combining filter nodes to make a tree is a bit tricky. Well, it would be nice with an environment extensively used for making filters, because it would simply make it easier and quicker to make them.

- **History Listener -** When trying to filter a concept, Conzilla first looks for a filter associated to the concept and if there is no one it looks in the current concept-map. But assume you are browsing a knowledge patch about mathematics and in a concept-map about geometry there is no filter ever associated with either the concepts or the map itself. Then it would be nice if the program could check with higher level maps if there are some filters to use. A possible solution to this problem is to introduce a History Listener that registers the user's path among the maps. Then Conzilla could back-trace this path to a find a fitting filter. One problem with History Listeners is that complex navigation among different knowledge patches could result in wrong filters being presented. Therefor you need quite advanced History Listeners, which perhaps will be too complex to design.

Appendix A

Glossary

A.1 Special terms

Here are some special terms used in this thesis.

aspect A *concept* can be divided up into different *aspects*. The content describing the concept will be filtered through these aspects.

concept A *concept* is a representation of a mental object.

concept-map Visualization of associated *concepts* and conceptual relationships. Has similarities with mind-maps.

content An explanation of a concept. It could be a document, a video, sound or something else that has learning potential and which it is possible to give reference to.

filter node A representation of an *aspect* in the model of filters.

filter tree A tree of connected *filter nodes***.**

knowledge patch A set of related concept-maps, covering the same area of knowledge.

metadata This is a group of data with information about a concept or a content component. It could describe who the author is, when it was created and some keywords that describe the object. These keywords are used when filtering the content.

A.2 Abbreviations

The following abbreviations are used in this thesis.

API (Application Programming Interface) - a set of classes and functions for performing certain tasks, while hiding the underlying details.

CID (Centre for user-oriented IT design) – an inter-disciplinary competence centre at KTH. Activities include a three-part collaboration between IT-industry, user

organizations and university researchers. For more information, see http://www.nada.kth.se/cid.

GUI (Graphical User Interface) – the interface to an application that the user sees, and which uses graphical elements such as buttons and menus for interaction.

IMS (Instructional Management Systems) – a global coalition of academic, commercial and government organizations, working together to define the Internet architecture for learning. For more information, see http://www.imsproject.org.

JDK (Java Development Kit) – a development environment for writing applets and applications that conform to the Java platform.

UML (Unified Modeling Language) – a language for specifying, visualizing, constructing and documenting the essences of software systems.

URI (Uniform Resource Identifier) – a URI is a string of characters for identifying an abstract or physical resource.

URL (Uniform Resource Locator) – a subset of URI that identifies resources via a representation of a primary access mechanism (e.g., the network locations).

XML (Extensible Markup Language) – a meta language that is used to define specialized markup languages (like HTML), which can be used to transmit data in a portable way.

Bibliography

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

[2] *Java in a Nutshell,* GNU

[3] *Conzilla – Towards a Concept Browser,* Nilsson M., Palmér M – Centre for user-oriented IT-Design (CID-53), KTH, Stockholm, Sweden, 1999

[4] *Conceptual Navigation and Multiple Scale Narration in a Knowledge Manifold,* Naeve A. – Centre for user-oriented IT-Design (CID-52), TRITA-NA-D9910, KTH, Stockholm, Sweden, 1999

[5] The Object Management Group, http://www.omg.org

[6] W3C - The World Wide Web Consortium, http://www.w3.org/

[7] The IMS Project, http://www.imsproject.org/

[8] Java™ Platform Documentation, http://java.sun.com/docs/

[9] A Visual Index to the Swing Components, http://java.sun.com/docs/books/tutorial/uiswing/components/components.html

[10] Swing API Specification, Swing API Specification, http://java.sun.com/products/jfc/swingdoc-api-1.1.1/index.html

[11] Conzilla home page, http://cid.nada.kth.se/il/conzilla/default.html

[12] The Conzilla Design, http://www.nada.kth.se/~mini/conzilla-design/conzilla-design.html

[13] *The Unified Modeling Language User Guide*, Booch G., Rumbaugh J., Jacobson I. – Addison Wesley Longman 1999

[14] UML Resource page, http://www.omg.org/uml