

Westfälische Wilhelms-Universität Münster

Englisches Seminar

Seminar: PS Textlinguistics

Lecturer: Dr. Florian Panitz

SS 2009

27.08.2009

COIFFEUR 2 — AN EXTENSIBLE AND USABLE  
APPROACH TO PRESENTING SHORT TEXTS  
USING EXTENDED SEMANTICAL MARKUP

Eugen Ruppert  
Oberschlesierstr. 5  
48151 Münster  
0251/7038791  
studium@euge.de  
Matrikel 343206

Language, Text and Information

---

# Contents

|          |                                                 |           |
|----------|-------------------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                             | <b>1</b>  |
| <b>2</b> | <b>XML</b>                                      | <b>1</b>  |
| 2.1      | Syntax . . . . .                                | 1         |
| 2.2      | The DTD . . . . .                               | 4         |
| 2.3      | Problems with XML . . . . .                     | 5         |
| <b>3</b> | <b>A different Approach</b>                     | <b>6</b>  |
| 3.1      | Semantic Markup . . . . .                       | 6         |
| 3.2      | The “truly” Semantic Approach . . . . .         | 8         |
| <b>4</b> | <b>Technical Description</b>                    | <b>9</b>  |
| 4.1      | Creation . . . . .                              | 10        |
| 4.2      | Representation . . . . .                        | 12        |
| <b>5</b> | <b>Results</b>                                  | <b>15</b> |
| <b>6</b> | <b>Conclusion</b>                               | <b>16</b> |
| <b>7</b> | <b>Outlook</b>                                  | <b>17</b> |
|          | <b>References</b>                               | <b>18</b> |
|          | <b>Appendix</b>                                 | <b>19</b> |
|          | A browser screenshot of the HTML file . . . . . | 19        |
|          | A printout of the HTML file . . . . .           | 22        |

# 1 Introduction

This term paper deals with the representation of short tagged texts in the web browser. I chose short texts — poetry in particular — for two reasons. The first one is that short texts can be analysed much faster, analysing a short story or even a novel would require much computation and therefore also time. The second reason is that poetry (or short prose) contains much more stylistic elements than longer texts and the means for analysing them are not the same.

I extended *Coiffeur* (Ruppert 2008) — a Perl script that I wrote for computational text analysis — to deal with user defined categories and classes. The texts are processed and put it into a `.html` file. The result can be embedded into other documents and because of a multitude of classes bundled with each word the user can write stylesheets (explained on p.7) to create his own visualisation schemes.

So, in this term paper I first present the XML (Extensible Markup Language) format which is often used by corpus linguists. I discuss the advantages and shortcomings of XML and present another approach, which is influenced by the “Semantic Web” idea.

After that I am going to give a technical explanation with instructions on how to create such representations. The technical explanation only covers the basic workings and does not explain the actual code of the extended *Coiffeur 2* because that would go beyond the scope of this paper. The results of the analysis of *Todesfuge* by Paul Celan are presented in the end, followed by the conclusion and a short outlook.

## 2 XML

### 2.1 Syntax

XML<sup>1</sup> is a Language used for markup of texts and data. With XML you can represent everything that is representable as a tree, since the XML format

---

<sup>1</sup> Extensible Markup Language, designed for quick development of formal document formats. These should be extensible and human-legible, so that you can pass the file to anyone and they can start adding new ‘objects’ or modify the existing ones without the need for special software (W3C 2008).

itself is a tree structure. The syntax is rather simple, the document consists of nested elements where every element has a start and an end tag. The start tag is the element name enclosed in angle brackets (`<element>`) and the end tag has a closing mark `'/'` before the element name (`</element>`).

**Listing 1:** The basic document structure

```

<rootelement>
  <childelement1>Content</childelement1>
  <childelement2>Content</childelement2>
  ...
5 </rootelement>

```

A common example for XML usage is a document containing customer information, such as name and address:

**Listing 2:** The first draft on the customer document (second address element omitted)

```

<?xml version='1.0' encoding='utf-8'>

<customers>
  <customer id='1'>
5     <firstname>John</firstname>
     <lastname>Miller</lastname>
     <address>
10        <street>Wellington Road</street>
        <housesnr>15</housesnr>
        <postcode>42153 </postcode>
        <city>Motown</city>
     </address>
  </customer>
15  <customer id='2'>
     <firstname>Susan</firstname>
     <lastname>Steinbeck</lastname>
     <address>
20        ...
     </address>
  </customer>
</customers>

```

Each customer has his own unique id, which can be combined with orders to keep track of who ordered what. You can edit the existing customers or add new ones without disturbing the structure of others. The next customer could be a company and *first name* and *last name* are not appropriate for a company. But you can extend your structure by saying that a customer either has a *first name* and *last name* or a *company name*. The address structure remains the same. Thus, the next customer is represented like this:

**Listing 3:** Extension of the first draft with different element structure (address omitted)

```

<customer id='3'>
  <companyname>ACME Industries</companyname>
  <address>
    ...
  </address>
</customer>

```

The problem now is that the information on the type of customer (person or company) is *implicit*, you can distinguish them by looking at the child elements of the customer element. To help your application it is better to make all information *explicit*. This can be done by adding further attributes to your elements. As we want to state whether a customer is a person or a company, therefore we include the attribute `type` into the customer element, so the final structure looks like this:

**Listing 4:** The final XML document containing customer data (some address elements omitted)

```

<?xml version='1.0' encoding='utf-8'>
<customers>
  <customer id='1' type='person'>
    <firstname>John</firstname>
    <lastname>Miller</lastname>
    <address>
      <street>Wellington Road</street>
      <housenr>15</housenr>
      <postcode>42153 </postcode>
      <city>Motown</city>
    </address>
  </customer>
  <customer id='2' type='person'>
    <firstname>Susan</firstname>
    <lastname>Steinbeck</lastname>
    <address>
      ...
    </address>
  </customer>
  <customer id='3' type='company'>
    <companyname>ACME Industries</companyname>
    <address>
      ...
    </address>
  </customer>
</customers>

```

This is the extensibility of XML: you can start with a simple document structure and extend it to be more complex when you need it without the

need of special programming skills that are needed for other document or data storage systems like databases.

## 2.2 The DTD

When you work with different people on a document you need to ensure that people do not extend the structure as they want to because this can lead to problems. Maybe your program needs certain elements to function properly or it cannot deal with the ones you added and thus ignores them. To prevent these problems there is usually a DTD (Document Type Definition) which states how the elements should be structured. In Listing 5 you see a commented DTD for the customer example from the previous section.

**Listing 5:** The DTD for customer documents

```

<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE customers [
<! ELEMENT customers      (customer)+ >
5  -- The root element "customers" contains one or more "customer" elements
   (indicated by the plus sign).  --

<! ELEMENT customer      (((firstname, lastname)|companyname), address) >
10 -- A "customer" element is built of either a "firstname and lastname" or a
   "companyname". An "address" element follows in both cases.  --

<! ATTLIST customer      id          #CDATA #REQUIRED
                           type      (person|company) #REQUIRED >
15 -- The "customer" element has two attributes, "id" and "type", both are
   required. The "id" attribute can be any string of characters, the "type"
   attribute is either "person" or "company".  --

<! ELEMENT firstname      #PCDATA >
<! ELEMENT lastname       #PCDATA >
<! ELEMENT companyname    #PCDATA >

20 <! ELEMENT address      (street, housenr, postcode, city) >
   -- The "address" element consists of "street", "housenr", "postcode"
   and "city".  --

<! ELEMENT street         #PCDATA >
25 <! ELEMENT housenr      #PCDATA >
<! ELEMENT postcode      #PCDATA >
<! ELEMENT city          #PCDATA >
   -- The other elements contain parsed character data, which means that they
   have no child elements but only text.  --
]>

```

With this DTD you can validate your documents; a good validator will give you detailed information about where your document is not compliant with the DTD, so that you can easily fix it and keep the program working.

### 2.3 Problems with XML

As described in the previous section, the XML format has many advantages. But while the format is easy to read, most elements are spelled out completely, it can be really tedious to write XML. Every element has an opening and closing tag and thus adding many new objects is repetitive and boring task. To use XML efficiently you need a custom written software that helps to work with the XML structure you designed. Commonly you also need to query the data, e.g. *list all customers that are companies* or *list all customers from the city Münster*, add new elements (preferably via a form that automatically tags the input values) and output your data, e.g. print the address on an envelope automatically. It should be clear that because you designed your own proprietary data structure you need your own software to work with it.

I consider this to be a major drawback, XML is not usable for everyday work. In the seminar on Textlinguistics, when we listened to a presentation on XML, one of the participants asked what you would need XML for. She could not see a reason why someone would perform such tedious work when there is no apparent real-world usage emerging from this.

I think that this is a legitimate question. You can create your highly tagged documents. But this data is not necessarily more useful for you when you do not have (written) the appropriate tools to view and modify it. Most text editors are suitable for writing and editing XML files, yet I am not aware of one program that lets you load *any* XML document and perform meaningful work with it.<sup>2</sup> And because of this XML is not very widespread among people with no Computer Science background.

Another problem is that you cannot just load the XML file and work with it. Usually web browsers are linked to `.xml` files. But when you open such a

---

<sup>2</sup> I think of a program that automatically validates the document according to the DTD, displays the data in a nice format (e.g. a single, editable form for each object) and is able to help you build queries as it *understands* the structure of your elements. It should also be able to perform the query and list the results in a useful manner.

file you are in the best case shown the indented tree view of the document.<sup>3</sup> You can search for a single instance of your desired word consecutively (no filtering) and you cannot edit the document.

**Figure 1:** The representation of XML documents in the browser (left: Firefox 3.0; right: Internet Explorer 7.0)

## 3 A different Approach

### 3.1 Semantic Markup

About the year 2000 the idea of “Semantic Markup” and the separation of content and style in HTML files emerged. Previous to that the documents self-contained all the styling information. When you wanted to write a heading you just embedded the text into a `<font>` or `<p>` (paragraph) element where you set the font size to a large value. Other elements were used because of their innate styling abilities, e.g. `<blockquote>` was used to indent a paragraph but the indented text did not have to be an actual quotation. Furthermore, elements like `<b>` or `<i>` were used to make the text bold or italic, without actually saying why the phrase had to be bold. The main element for page layout was the `<table>` element. The table was not used as an actual table but to layout the page in columns. Thus, the elements were chosen according to their looks and not to their function. This

<sup>3</sup> Earlier versions of Firefox even refused to display the structure, saying that there was no styling information linked to the document.

is still used today, but luckily most designers have started to create valid, well-structured and semantically meaningful documents.

The markup according to the look of the the elements poses many problems. Search engines do not distinguish between font-sizes and colours, they only analyse the element structure. Thus, they have difficulties in finding out what the text is about when there is no heading element. Users with mobile phones or vision impairment have difficulties in viewing the pages because their browsers or screen readers do not support the styling information of the `<font>` tag. They cannot navigate easily in the documents. When there are no headings explicitly stated, the reader software cannot perform certain functions, such as *skip to the next heading*. Also, when every text element contains its own styling information, the file size gets very large.

With the introduction and success of CSS<sup>4</sup> most problems were resolved. CSS was introduced to perform the separation of content and styling information. The HTML document should only contain data, elements can have their own ids or belong to a class of elements.<sup>5</sup> In the stylesheet there is styling information, such as colour or text size, linked to elements, classes or ids.

The main advantage of this separation is that you do not need to edit your documents when you want to change their look, you edit the stylesheet. And because many documents can share the same stylesheet, it enables the document creator to maintain a consistent look across all our his documents without editing every document, which can be very much work for a large website.

I am not going to present the CSS selectors and syntax in detail, but I show a few examples to demonstrate the simple, self-explaining syntax:

**Listing 6:** Some example CSS definitions

```
/* sets the background-color of the document "body" to blue */
body {
    background-color: blue;
}
5 /* sets the text-color of the element with the id "searchbar" to red */
```

---

<sup>4</sup> Cascading Style Sheets, the specifications for CSS level 1 were completed in 1996 (W3C 1996).

<sup>5</sup> For the specification of HTML 4.0 consult W3C (1999).

```
#searchbar {  
    text-color: red;  
}  
  
/* sets the font of the class "example" to italic */  
.example {  
    font-style: italic;  
}
```

CSS is easy to maintain and therefore most designers switch to using it and tag their documents semantically. But when I heard of “Semantic Markup” I thought that you can drive this idea further and insert much more information into your documents, the semantic information of the words and phrases you use. You could include information on the gender of the participants, parts-of-speech tags and even stylistic elements, such as metaphor or alliteration. When you make it visible the result will be a *truly semantic markup*. This is what I try to achieve.

### 3.2 The “truly” Semantic Approach

As shown before, there is no *general purpose* program for XML, and web browsers are not suited for pure XML. But they are able to display HTML (HyperText Markup Language) documents and HTML and XML have a common ancestor: SGML (Standard Generalized Markup Language), which means that they share a very similar structure.<sup>6</sup>

Therefore I chose XHTML as the basis for my approach. My aim is to display tagged poetry in the browser. This enables a wide user base to interact with the resulting hypertext documents. The user can select information he or she wants to view and the browser highlights the according words or phrases with the help of CSS. The style selection and switching is performed by JavaScript.

---

<sup>6</sup> Especially XHTML has the same markup conventions as XML:

- all elements must be closed
- all element names are spelled out in lower case
- empty elements need to be closed (like `<br />` or `<img />`).

These strict conventions make it easier to program structured documents without ambiguities and therefore I prefer it over the rather loose HTML syntax.

Thus, for my proposed format you only need a browser which is capable of using CSS and JavaScript – the default configuration of every graphical browser. And even with disabled CSS and JavaScript<sup>7</sup> you can still view and print the (unstyled) document, it looks fine but lacks the possibility of highlighting special elements in the text.

## Paul Celan – Todesfuge (1947)

Schwarze Milch der Frühe wir trinken sie abends  
 wir trinken sie mittags und morgens wir trinken sie nachts  
 wir trinken und trinken  
 wir schaufeln ein Grab in den Lueften da liegt man nicht eng  
 Ein Mann wohnt im Haus der spielt mit den Schlangen der schreibt  
 der schreibt wenn es dunkelt nach Deutschland dein goldenes Haar Margarete  
 er schreibt es und tritt vor das Haus und es blitzen die Sterne er pfeift seine Rueden herbei  
 er pfeift seine Juden hervor läßt schaufeln ein Grab in der Erde  
 er befiehlt uns spielt auf nun zum Tanz

Schwarze Milch der Frühe wir trinken dich nachts  
 wir trinken dich morgens und mittags wir trinken dich abends  
 wir trinken und trinken  
 Ein Mann wohnt im Haus der spielt mit den Schlangen der schreibt  
 der schreibt wenn es dunkelt nach Deutschland dein goldenes Haar Margarete  
 Dein aschenes Haar Sulamith wir schaufeln ein Grab in den Lueften da liegt man nicht eng

Figure 2: A screenshot of the result with disabled CSS

I have thought about implementing my approach with a fall-back that would come into play when JavaScript is disabled. But it would require work on the server side, with the implication that you cannot just edit your files and view them on your computer. Instead, you would need a PHP-enabled<sup>8</sup> web server and this is against my intent to make the documents as easily viewable and distributable as possible. In the current form they can be viewed on the local computer and shared by sending them via email.

## 4 Technical Description

My approach consists of two distinct parts, representation of tagged texts and their creation. Before I start explaining either of them I present the transformation process. The text from Listing 7 is transformed into HTML

<sup>7</sup> JavaScript is considered to be a security risk and therefore could be turned off in the browser.

<sup>8</sup> PHP: Hypertext Preprocessor; with PHP you are able to run code when the document is requested.

code (Listing 8).<sup>9</sup> Each stanza, line and word are counted and presented with a unique id. Furthermore, each word belongs to a list of classes. These and the number of occurrence are put into the title attribute, the user can “hover” the mouse over the word to view the classes that have been applied to it.

**Listing 7:** The example text

```
Schwarze Milch der Frühe wir trinken sie abends
```

**Listing 8:** The resulting HTML code

```
<p id="stanza1" class="stanza oddstanza">

  <span id="line1" class="line oddline">
    <span class=" Metapher1 Adjektiv Oxymoron1 schwarz boese occ-class0 occ-
      class-max" id="word1" title="1 occurrence; Classes: Metapher1 Adjektiv
      Oxymoron1 schwarz boese occ-class0 occ-class-max">Schwarze</span>
5    <span class=" Nomen Metapher1 Oxymoron1 boese occ-class0 occ-class-max" id
      ="word2" title="1 occurrence; Classes: Nomen Metapher1 Oxymoron1 boese
      occ-class0 occ-class-max">Milch</span>
    <span class=" Metapher1 Oxymoron1 Artikel occ-class0 occ-class-max" id="
      word3" title="1 occurrence; Classes: Metapher1 Oxymoron1 Artikel occ-
      class0 occ-class-max">der</span>
    <span class=" Nomen Metapher1 Oxymoron1 morgens occ-class0 occ-class-max"
      id="word4" title="1 occurrence; Classes: Nomen Metapher1 Oxymoron1
      morgens occ-class0 occ-class-max">Frühe</span>
    <span class=" Geschlecht-unbestimmt Pronomen occ-class0 occ-class-max" id=
      "word5" title="1 occurrence; Classes: Geschlecht-unbestimmt Pronomen
      occ-class0 occ-class-max">wir</span>
    <span class=" Verb occ-class0 occ-class-max" id="word6" title="1
      occurrence; Classes: Verb occ-class0 occ-class-max">trinken</span>
10    <span class=" Pronomen occ-class0 occ-class-max" id="word7" title="1
      occurrence; Classes: Pronomen occ-class0 occ-class-max">sie</span>
    <span class=" abends Adverb occ-class0 occ-class-max" id="word8" title="1
      occurrence; Classes: abends Adverb occ-class0 occ-class-max">abends</
      span>
    </span>
  </p>
```

## 4.1 Creation

It is apparent from the Listings above that it can be very tedious to manually annotate even a short text. You have to enclose every word in a `<span>`

<sup>9</sup> The execution of the script is performed with the command “perl coiffeur2.pl textfile.txt”, for my analysis it was “perl coiffeur2.pl todesfuge.txt”. The operating system needs to be able to execute Perl scripts.

element and then assign different classes to it. Especially syntactic annotation (article, pronoun) is repetitive, boring and error-prone.

Therefore I decided to write a script that would automatically annotate the words based on (1) word lists and (2) computational text analysis. The word lists are in a single text file (`classes.txt`) and look like the following Listing:

**Listing 9:** A short example class list

```

noun (woman, man, store, cat)
pronoun (he, she, it, we, they, this)
article (a, the)
verb (go, run, sell, jump, read, kill)
5 adjective (gold.*, blue)

female (she, woman, girl)
male (he, man, boy)

10 colour (blue, red, gold, black, blue)

oxymoron (black milk)
metaphor (heart is burning, black milk)

```

These lists are extensible, every user can edit the file, add new categories and add words to existing ones. When the script is executed, it reads the list and applies the according classes to each word. It is designed to allow multiple classes per word because usually there is syntactical information for every word bundled with various semantical and statistical classes.

A special feature is that you are not limited to use single words for the classes. There are often multiple words that combine to build a rhetorical figure, such as the oxymoron *black milk*.<sup>10</sup>

Another speciality is that you can use Regular Expressions in the word lists to keep them short and clear. The expression *gold.\** from the Listing above matches all words that start with *gold*, like the adjectives *gold* or *golden*. This is especially helpful with German Language because it has many inflections. You have to be careful, though, to prevent “false positives”. *Gold.\** would also match *gold-rush* which is not an adjective.

<sup>10</sup> For the analysis of *Todesfuge*, I put complete stanzas from the poem to mark the fugue structure. It is possible to put the whole poem into the classes file to mark every word with a certain class. But punctuation marks and other non-word characters should be removed because they are used to structure the classes file (compare Listing 9) and could lead to incorrectly tagged texts.

The computational text analysis is performed automatically. It divides the text into paragraphs, the lines into verses and counts every paragraph, line and word. It is also able to look for repetitions of words or phrases and perform basic stylistic analysis, such as alliteration or anaphora.

## 4.2 Representation

The representation is performed by HTML and CSS. Each word of the text lies inside a `<span>` element. `<span>` elements are *inline* elements which can be added at random to a text without causing any side effects, such as line breaks, format changes or other display effects (compare Figure 2). Each `<span>` element belongs to multiple classes which can be styled individually.

In my design, each stylesheet actually consists of two individual files, one for foreground styling and one for the background. When there are foreground and background styles for every class, the user can combine them to discover interrelations (e.g. between the syntax and rhetorical figures).

**Listing 10:** The foreground stylesheet for the polarity “good – bad”

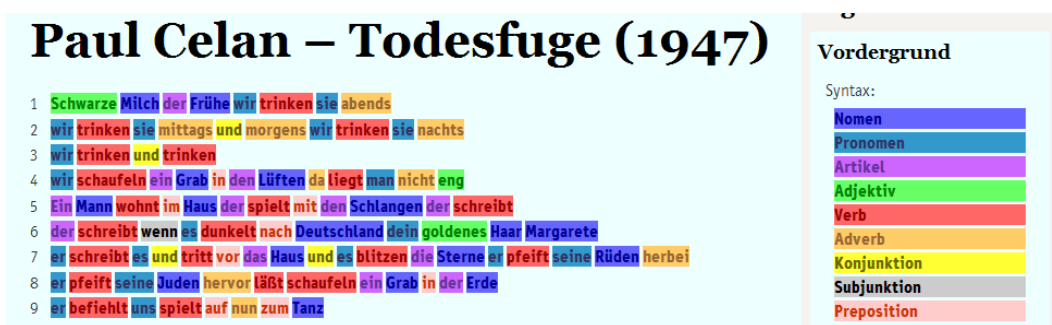
```
.front .good-bad {  
    display:block;  
}  
5 .gut {  
    color: #3cb11d;  
    font-weight:bold;  
}  
10 .boese {  
    color: #5b1f77;  
    font-weight:bold;  
}
```

**Listing 11:** The foreground stylesheet for the polarity “good – bad”

```
.back .good-bad {  
    display:block;  
}  
5 .gut {  
    background-color: #a3f78c;  
}  
.boese {  
    background-color: #db94fc;  
}
```

There are two classes styled in these files, *gut* and *boese*. The foreground stylesheet sets a rather dark text colour and the font-weight to bold. The

background stylesheet sets the background colour to a lighter colour. Both colours are similar in hue, but because of the difference in lightness one can distinguish the text even if both styles are activated at the same time (compare Figure 3).<sup>11</sup> For this particular implementation there is a style instruction at the top in both stylesheets. It is used to display the legend when the according style is selected. The legends are in <div> containers with the classes *front* and *back* (cf. Listing 12). The default stylesheets hides them with the instruction `display:none;`. When another stylesheet is activated, it overwrites the display instruction for this particular class so that only the legends of the activated stylesheets are displayed.



**Figure 3:** The first stanza with “syntax stylesheets” activated for the foreground and background

**Listing 12:** An extract from main HTML file, showing the legend items for the polarity “good – bad”

```

<div class="front">
  <ul class="good-bad">
    <li class="gut">gut</li>
    <li class="boese">böse</li>
  </ul>
</div>

```

The user can activate various stylesheets to view structural, semantical or stylistic information. The selectors are hyperlinks <a> with a `rel` attribute. In this attribute the path to the according stylesheet is specified. The foreground stylesheets are put into the *style1* <div> container, for the background *style2* is used (Listing 13).

<sup>11</sup> There is a speciality with anaphora. The script checks for both stanza anaphora and line anaphora because it makes a difference whether multiple stanzas start with the same word or there are anaphora in the stanzas. Therefore, it can happen that the first word of a stanza belongs to both the *anaphline* and the *anaphstanza* classes. For this occasion the *anaphstanza* class contains different style information so that it does not interfere with the style of *anaphstanza*.

Listing 13: The style selectors (compare with the representation of *Todesfuge*)

```

<h3>Auswahl von Auszeichnungen</h3>
<div id="style1">
  <h4>Vordergrund</h4>
  <ul>
    <li><a href="#" rel="css/nostyle.css">Keine Auszeichnung</a></li>
  </ul>

  <h5>Struktur:</h5>
  <ul>
    <li><a href="#" rel="css/syntax.css">Syntax</a></li>
    <li><a href="#" rel="css/rhyme.css">Reim/Asonanz</a></li>
    <li><a href="#" rel="css/fugue.css">Fugenschema</a></li>
    <li><a href="#" rel="css/lines.css">Verse</a></li>
    <li><a href="#" rel="css/rep-class.css">Wortwiederholungen</a>
  </li>
  </ul>
</div>

<div id="style2">
  <h4>Hintergrund</h4>
  <ul>
    <li><a href="#" rel="css/nostyle-back.css">Keine Auszeichnung</a></li>
  </ul>

  <h5>Struktur:</h5>
  <ul>
    <li><a href="#" rel="css/syntax-back.css">Syntax</a></li>
    <li><a href="#" rel="css/rhyme-back.css">Reim/Asonanz</a></li>
    <li><a href="#" rel="css/fugue-back.css">Fugenschema</a></li>
    <li><a href="#" rel="css/lines-back.css">Verse</a></li>
    <li><a href="#" rel="css/rep-class-back.css">
      Wortwiederholungen</a></li>
  </ul>
</div>
</div>

```

The layout and function of the document were tested in Internet Explorer 7 and 8 and in Firefox 3. I have also created a print stylesheet which hides unnecessary information and displays the legend horizontally (Figure 4).



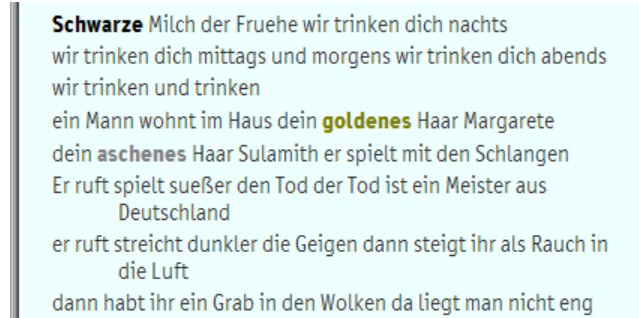
Figure 4: The print preview in Firefox

## 5 Results

The results can be found on my website at <http://euge.de/todesfuge/>.<sup>12</sup> I analysed the poem *Todesfuge* by Paul Celan and enriched it with background information, media and links to other relevant websites. This is the way I think poems should be presented to enable easy analysis. It is good if you are able to find every structural or stylistic element by yourself, my result does not try to undermine it. But I think with the proper visual representation you are able to write a more accurate analysis and better understand how the writer achieves his effects. You can see it when you view the fugue scheme in the poem. It starts rather slow and ordered and moves in circles (Dux – Comes). But at the end the structure dissolves and Dux and Comes mix together in line 28.

Some of the automatically inserted information can be questionable for some users. The check for alliteration is performed only by looking at the words and not line breaks or any other symbol. Thus, the last word of line 34, *Deutschland*, and the first of line 35 (*dein*) are marked as alliteration. I think this is acceptable and when some users prefer it differently, they can

<sup>12</sup> This is the HTML result. An archive containing the complete package (example files, the Perl script and the complete website) is available at:  
<http://shorty.euge.de/program/coiffeur2/>.

A screenshot of a web browser window showing a poem. The text is displayed in a light blue background with a vertical line on the left side. The text is as follows:

**Schwarze** Milch der Fruehe wir trinken dich nachts  
wir trinken dich mittags und morgens wir trinken dich abends  
wir trinken und trinken  
ein Mann wohnt im Haus dein **goldenes** Haar Margarete  
dein **aschenes** Haar Sulamith er spielt mit den Schlangen  
Er ruft spielt sueßer den Tod der Tod ist ein Meister aus  
Deutschland  
er ruft streicht dunkler die Geigen dann steigt ihr als Rauch in  
die Luft  
dann habt ihr ein Grab in den Wolken da liegt man nicht eng

**Figure 5:** The Internet Explorer 7 does not display line numbers

remove unwanted classes by hand, it is still much faster than annotating these classes manually.

The lines are indented left, so that when a line break occurs in a line it is still noticeable that this is a continuing and not a new line. Line numbering is performed by CSS counters in the browser. Therefore, the browser must support counters. If it does not—like Internet Explorer 7—the viewer just sees the text (Figure 5). An advantage over manual line numbering is that you can now copy parts of the poem and when you paste them somewhere else, all of the numbers have disappeared and do not interfere with further editing.

## 6 Conclusion

There are of course limitations to my approach. When you view information in your browser there is no way to edit it. As with every XML file you need an editor to perform changes. Furthermore you can view selected information, yet it is not possible to form queries. You need to look at the displayed representation and find for yourself where certain elements correspond. But the visual representation is far better than trying to figure things out while looking at the `<span>` tags from Listing 8.

## 7 Outlook

The system that was presented in this paper is well suited for everyday usage and education. But there are already many tagged texts.<sup>13</sup> These can be transformed into the format that I propose. XSLT (Extensible Stylesheet Language Transformations) is a tool to transform XML documents into XML documents. Thus, it is possible to transform the documents created by *Coiffeur 2* into XML documents and edit them with XML-enabled software. The other way works too, because the information in my document format is explicit, even though the structure is not as deep as it would be in an XML corpus. Through this data interoperability there are benefits for both sides, *Coiffeur 2* is one of the fastest way to tag texts and having tagged texts from different sources enriched with styling information is beneficiary to many users.

A further point is that the script remains extensible, you can insert your own functions into the process, e.g. a part-of-speech tagger or other analytical tools. As with every program, there is much room for improvement.

And while it is rather simple to write new style sheets for your own classes, it can be hard to get the right colour codes into the styles. A possible solution for this are CSS Variables (Glazman and Hyatt 2008). They currently only exist as a draft, but would make this application much easier. Many colour pairs (dark – light) would be defined in the main stylesheet and the class stylesheets could select suitable ones without having to provide the colours themselves. This would save time and keep a consistent look between different documents.

---

<sup>13</sup> For example, Wordhoard contains all the plays written by Shakespeare, enriched with much information (Northwestern University 2008).

## References

Glazman, D. and D. Hyatt (2008). CSS Variables. Website.

URI: <http://disruptive-innovations.com/zoo/cssvariables/>.

Invent Media (2008). Invent Media Flash Inline MP3 Player. Website.

URI: <http://www.inventmedia.com.au/flashmp3.html>.

Northwestern University (2008). WordHoard. An application for the close reading and scholarly analysis of deeply tagged texts. Website and Software.

URI: <http://wordhoard.northwestern.edu/userman/>.

Ruppert, E. (2008). Coiffeur — Computational Text Analysis in Perl. Website.

URI: <http://shorty.euge.de/program/coiffeur>.

W3C (1996). Cascading Style Sheets, level 1. Website.

URI: <http://www.w3.org/TR/CSS1/>.

W3C (1999). HTML 4.01 Specification. Website.

URI: <http://www.w3.org/TR/html401/>.

W3C (2007). Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. Website.

URI: <http://www.w3.org/TR/2007/CR-CSS21-20070719/>.

W3C (2008). Extensible Markup Language (XML) 1.0. Website.

URI: <http://www.w3.org/TR/REC-xml/>.

All of the mentioned websites and online documents were available at the point of writing this essay.

# Appendix

Diese Representation der Todesfuge von Paul Celan entstand im Rahmen einer Hausarbeit zur Semantischen Annotation und Representation von Gedichten (und anderen kurzen Texten). Die Arbeit erweitert das computerlinguistische Tool [Coiffeur](#) um selbstgewählte Klassen, die den Wörtern hinzugefügt werden.

Sie können sowohl [die PDF-Datei von Coiffeur 2](#) als auch [das komplette Paket](#) herunterladen. Feedback (Kommentare, Fehlermeldungen) ist [ausdrücklich erwünscht](#).

## Paul Celan – Todesfuge (1947)

1 **Schwarze Milch der Frühe** wir trinken sie abends  
2 wir trinken sie mittags und morgens wir trinken sie nachts  
3 wir trinken und trinken  
4 wir schaufeln ein **Grab in den Lüften** da liegt man nicht eng  
5 Ein Mann wohnt im Haus der **spielt mit den Schlangen** der schreibt  
6 der schreibt wenn es dunkelt nach Deutschland dein goldenes Haar Margarete  
7 er schreibt es und tritt vor das Haus und es blitzen die Sterne er pfeift seine Rüden herbei  
8 er pfeift seine Juden hervor läßt schaufeln ein Grab in der Erde  
9 er befiehlt uns spielt auf nun zum Tanz

10 **Schwarze Milch der Frühe** wir trinken dich nachts  
11 wir trinken dich morgens und mittags wir trinken dich abends  
12 wir trinken und trinken  
13 Ein Mann wohnt im Haus der **spielt mit den Schlangen** der schreibt  
14 der schreibt wenn es dunkelt nach Deutschland dein goldenes Haar Margarete  
15 Dein aschenes Haar Sulamith wir schaufeln ein **Grab in den Lüften** da liegt man nicht eng

16 Er ruft stecht tiefer ins Erdreich ihr einen ihr andern singet und spielt  
17 er greift nach dem Eisen im Gurt er schwingts seine Augen sind blau  
18 stecht tiefer die Spaten ihr einen ihr andern spielt weiter zum Tanz auf

19 **Schwarze Milch der Frühe** wir trinken dich nachts  
20 wir trinken dich mittags und morgens wir trinken dich abends  
21 wir trinken und trinken  
22 ein Mann wohnt im Haus dein goldenes Haar Margarete  
23 dein aschenes Haar Sulamith er **spielt mit den Schlangen**  
24 Er ruft spielt süßer den Tod der Tod ist ein Meister aus Deutschland  
25 er ruft streicht dunkler die Geigen dann steigt **ihr als Rauch** in die Luft  
26 dann habt ihr ein **Grab in den Wolken** da liegt man nicht eng

27 **Schwarze Milch der Frühe** wir trinken dich nachts  
28 wir trinken dich mittags der **Tod ist ein Meister** aus Deutschland  
29 wir trinken dich abends und morgens wir trinken und trinken  
30 der **Tod ist ein Meister** aus Deutschland sein Auge ist blau  
31 er trifft dich mit bleierner Kugel er trifft dich genau  
32 ein Mann wohnt im Haus dein goldenes Haar Margarete  
33 er hetzt seine Rüden auf uns er schenkt uns ein **Grab in der Luft**  
34 er **spielt mit den Schlangen** und träumet der **Tod ist ein Meister** aus Deutschland

35 dein goldenes Haar Margarete  
36 dein aschenes Haar Sulamith

### Vortrag

Ein Originalvortrag des Gedichts vom Autor. Zum Abspielen auf den «Play-Knopf» klicken.

**von Gedi:**  
.....

### Legende

#### Vordergrund

**Metapher: Schwarze Milch der Frühe**  
**Metapher: Spiel mit den Schlangen**  
**Metapher: Grab in den Lüften**  
**Vergleich: ihr als Rauch**

#### Hintergrund

Fugenschema:  
Dux  
Comes

### Auswahl von Auszeichnungen

#### Vordergrund

Keine Auszeichnung

#### Struktur:

Syntax  
Reim/Asonanz  
Fugenschema  
Verse  
Wortwiederholungen

#### Semantische Eigenschaften und Wortketten:

Farben  
Geschlecht  
Musik – Arbeit  
Luft – Erde  
Gut – Böse  
Tageszeiten

#### Stilmittel:

Anapher  
Alliteration  
Wiederholung  
Metapher/Vergleich  
Oxymoron  
Parallelismus

#### Hintergrund

Keine Auszeichnung

#### Struktur:

Syntax  
Reim/Asonanz  
Fugenschema  
Verse  
Wortwiederholungen

Figure 6: Browser screenshot (part 1 of 3)

**Semantische Eigenschaften und Wortketten:**

- Farben
- Geschlecht
- Musik – Arbeit
- Luft – Erde
- Gut – Böse
- Tageszeiten

**Stilmittel:**

- Anapher
- Alliteration
- Wiederholung
- Metapher/Vergleich
- Oxymoron
- Parallelismus

## Weiteres Material

### Informationen zum Autor

Aus der [Wikipedia](#):

Paul Celan wurde in Czernowitz, der Hauptstadt der Bukowina in Nordrumänien in einer deutschsprachigen jüdischen Familie geboren. Er war der einzige Sohn von Leo Antschel-Teitler (\* 1890 in Schipenitz bei Czernowitz) und dessen Ehefrau Fritzi, geb. Schrager (\* 1895 in Sadagora); erste Wohnung in der Wassilkogasse in Czernowitz.

Celan besuchte zunächst die deutsche, dann die hebräische Grundschule, fünf Jahre das Rumänische Staatsgymnasium und bis zum Abitur am 3. Juni 1938 das Ukrainische Staatsgymnasium. Im selben Jahr begann er ein Medizinstudium in Tours, kehrte jedoch ein Jahr später nach Rumänien zurück, um dort Romanistik zu studieren. 1940 wurde die nördliche Bukowina und somit auch Celans Heimatstadt Czernowitz von der Sowjetunion besetzt. Celan konnte sein Studium zunächst fortsetzen. Als jedoch 1941 rumänische und deutsche Truppen Czernowitz besetzten, wurden die Juden in ein Ghetto gezwungen. Celans Eltern wurden 1942 deportiert. In einem Lager in Transnistrien starb sein Vater an Typhus, seine Mutter wurde erschossen.

Von 1942 bis 1943 wurde Celan in verschiedenen rumänischen Arbeitslagern festgehalten und musste Zwangsarbeit im südmdolauischen Straßenbau leisten. Nach der Befreiung (Czernowitz wurde August 1944 von der Roten Armee eingenommen) kehrte Celan im Dezember 1944 nach Czernowitz zurück und nahm sein Studium wieder auf. 1947 ging er über Ungarn nach Wien und siedelte 1948 nach Paris über. Noch im selben Jahr erschien in Wien mit *Der Sand aus den Urnen* sein erster Gedichtband, der zunächst keine Beachtung fand.

Celan wurde mehrmals in psychiatrische Kliniken eingewiesen, z. B. vom 28. November 1965 bis 11. Juni 1966, weil er in einem Wahnzustand Lestrange mit einem Messer töten wollte. Im November 1967 entschied er und seine Frau, getrennt voneinander zu wohnen. Sie blieben aber weiterhin in Verbindung.

Die Umstände und das Datum von Celans Tod sind nicht geklärt. Vermutlich am 20. April 1970 suchte er den Freitod in der Seine am Pont Mirabeau. Celans Leichnam wurde am 1. Mai 1970 bei Courbevoie, zehn Kilometer flussabwärts von Paris, aus der Seine geborgen. Er wurde am 12. Mai 1970 auf dem Friedhof Thiais/Val-de-Marne beigesetzt. An diesem Tag starb Nelly Sachs, mit der er freundschaftlich verbunden war.

### Fuge

Aus der [Wikipedia](#):

Die Fuge (von lateinisch fuga = „Flucht“) ist ein musikalisches Kompositionsprinzip, das durch eine besondere Anordnung von Imitationen gekennzeichnet ist.

Besonderes Kennzeichen der Fuge ist ihre komplexe Themenverarbeitung. Eine Fuge beginnt mit der Exposition der Stimmen: Die erste Stimme trägt das prägnante, kurze Thema vor. Dieser Themenéinsatz wird auch als Dux (lat. „Führer“) bezeichnet. Hierzu gesellt sich eine zweite Stimme, die das Thema nun als Comes (lat. „Gefährte“) auf die Oberquinte (bzw. Unterquarte) versetzt vorträgt.

Figure 7: Browser screenshot (part 2 of 3)

## ER von Immanuel Weissglas

Bereits sechs Monate vor der Todesfuge schrieb der rumänische Dichter und Schulfreund Celans Immanuel Weissglas ein Gedicht, das für die Entstehung der Todesfuge Pate stand:

- 1 Wir heben Gräber in die Luft und siedeln
- 2 Mit Weib und Kind an dem gebotnen Ort.
- 3 Wir schaufeln fleißig, und die andern fiedeln,
- 4 Man schafft ein Grab und fährt im Tanzen fort.
  
- 5 ER will, daß über diese Därme dreister
- 6 Der Bogen strengere wie sein Antlitz streicht:
- 7 Spielt sanft vom Tod, er ist ein deutscher Meister,
- 8 Der durch die Lande als ein Nebel schleicht.
  
- 9 Und wenn die Dämmerung blutig quillt am Abend,
- 10 Öffn' ich nachzehend den verbissnen Mund,
- 11 Ein Haus für alle in die Lüfte grabend:
- 12 Breit wie der Sarg, schmal wie die Todesstund.
  
- 13 ER spielt im Haus mit Schlangen, dräut und dichtet,
- 14 In Deutschland dämmert es wie Gretchens Haar.
- 15 Das Grab in Wolken wird nicht eng gerichtet:
- 16 Da weit der Tod ein deutscher Meister war.

## Weiterführende Links

- Paul Celan – Die Todesfuge auf [celan-projekt.de](http://celan-projekt.de)
- [Unterrichtsmaterialien zur Todesfuge \(PDF\)](#) von Leo Koch
- Der Wikipedia-Eintrag zu [Todesfuge](#)

Eugen Ruppert 2009

Valid XHTML & CSS:



Figure 8: Browser screenshot (part 3 of 3)







---

Die vorliegende Arbeit “COIFFEUR 2—AN EXTENSIBLE AND USABLE APPROACH TO PRESENTING SHORT TEXTS USING EXTENDED SEMANTICAL MARKUP” umfasst etwa 3800 Wörter.<sup>14</sup>

Ich versichere hiermit, dass ich sie selbstständig und ohne fremde Hilfe verfasst habe, keine anderen Quellen und Hilfsmittel als die angegeben benutzt habe und dass die Stellen der Arbeit, die anderen Werken — auch elektronischen Medien — dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

Eugen Ruppert

---

<sup>14</sup> Wörterzählung mit ‘LaTeX word count’:  
<http://folk.uio.no/einarro/Services/texcount.html>