Submitted for Track 2 Knowledge Management

Klaus Prätor
Berlin-Brandenburg Academy of Sciences and the Humanities
Jägerstr. 22-23, 10117 Berlin
praetor@bbaw.de

Logic for Critical Editions

The application of declarative programming in the field of critical or scientific editions is to be seen in the larger frame of publication and, as we deal with electronic editions, in the context of Web and mark-up languages. But critical editions own specific attributes and difficulties. My aim is to show that techniques of logic programming are especially apt for the structures and problems of this domain.

Characteristics of critical editions

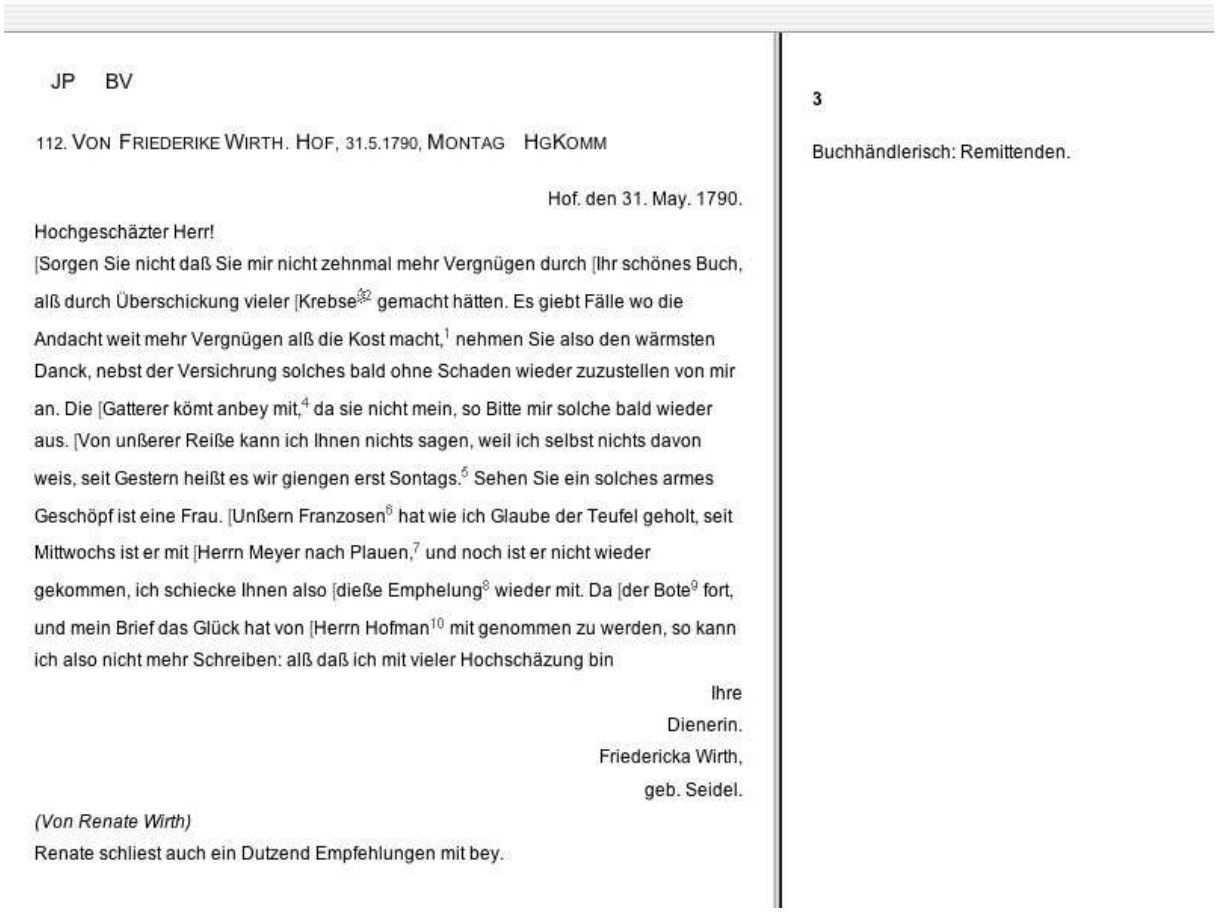
Critical editions are notoriously difficult sorts of text, as they form not one linear text but rather a complex of different variants and readings of a text, which are to be handled within a critical apparatus (think of a set of foot- or endnotes). They are enriched by commentaries with historical and philological information and made accessible by indices and directories. Other peculiarities are the mostly large extent of the editions and the fact that the period of production as well as of usage is very long reaching from decades to centuries. Especially in electronic editions it is necessary to care for sustainable availability and usability.

Some problems are due to the print form. Lack of space leads to elaborate systems of abbreviations and to steady considerations, which material and information can still be incorporated and which one has to be left out. The print can hardly handle the inherent nonlinearity of the documents and of course there is no thought of adaptation to different situations of usage. Compared to the print form electronic editions show some advantages, which result from the advanced ways of navigation and retrieval and from the possibility to provide different types of output. The larger storage capacities provide room for additional information regarding e.g. involved persons or historical circumstances. Hypertext capacities facilitate the constitution of temporal, spatial or thematic relations. Different editions may be nested and entirely new information spaces may be created in this way.

Pilot project Jean Paul

The Berlin-Brandenburg Academy of Sciences and Humanities is the home of thirty edition projects of all ages - editions in a broader sense including aside from work editions also source editions and dictionaries. Most of them are still focussed on the print form, but we are working on the migration to genuine electronic editions. As a pilot project we have chosen the

correspondence of Jean Paul, in the times of Goethe a much read and appreciated author of novels. We had two aims: to produce an electronic equivalent of the just finished first volume of the letters to Jean Paul and to demonstrate in an enriched selection some additional features of an electronic edition.



JP BV

112. VON FRIEDERIKE WIRTH. HOF, 31.5.1790, MONTAG HGKOMM

Hof. den 31. May. 1790.

Hochgeschäzter Herr!

[Sorgen Sie nicht daß Sie mir nicht zehnmal mehr Vergnügen durch [Ihr schönes Buch, alß durch Überschickung vieler [Krebse[2] gemacht hätten. Es giebt Fälle wo die Andacht weit mehr Vergnügen alß die Kost macht,[1] nehmen Sie also den wärmsten Danck, nebst der Versichrung solches bald ohne Schaden wieder zuzustellen von mir an. Die [Gatterer kömt anbey mit,[4] da sie nicht mein, so Bitte mir solche bald wieder aus. [Von unßerer Reiße kann ich Ihnen nichts sagen, weil ich selbst nichts davon weis, seit Gestern heißt es wir giengen erst Sontags.[5] Sehen Sie ein solches armes Geschöpf ist eine Frau. [Unßern Franzosen[6] hat wie ich Glaube der Teufel geholt, seit Mittwochs ist er mit [Herrn Meyer nach Plauen,[7] und noch ist er nicht wieder gekommen, ich schiecke Ihnen also [dieße Emphelung[8] wieder mit. Da [der Bote[9] fort, und mein Brief das Glück hat von [Herrn Hofman[10] mit genommen zu werden, so kann ich also nicht mehr Schreiben: alß daß ich mit vieler Hochschäzung bin

Ihre
Dienerin.
Friedericka Wirth,
geb. Seidel.

(Von Renate Wirth)

Renate schliest auch ein Dutzend Empfehlungen mit bey.

3

Buchhändlerisch: Remittenden.

To give an impression of the tasks and problems of the edition we look at a tiny part of the letter: the footnote number (3) on the right side within the commentary. It is not very important, may be only a habit from print times. On the surface there is this digit 3. One layer deeper, which is in this example the XML-Code to be envisioned by the browser, it looks like this:

- **presentation layer (XML)**
  <note id="3">Buchhändlerisch: Remittenden.</note>
- **+ style sheet** (**CSS)**
  note:before {content:attr(id);font-weight:bold;
  display:block; padding-top:10pt;padding-bottom:10pt;}

Here the number is no entity of its own, but only the value of an attribute, and it is displayed by a very special feature of CSS, which places some text before the respective tag, in our case the content is taken from the tag's attribute id. For a number of reasons this layer is not the one which will be the basis of the edition. Maybe we change our mind and want to display the commentary as a pop up. This should always be possible, especially regarding the long timeframe, in which a critical edition should be usable. Therefore there is a more fundamental layer :

- **archive layer (XML)**

<notep><pg>202</pg>
6
<jpzit>Krebse]</jpzit> Buchhändlerisch: Remittenden.</notep>

This code is the one, which will be archived and function as basis for the production of different presentation layers. At this level the footnote number does not exist at all. Instead we see tagged references to page and line of the print version as well as the lemma (Krebse) to which the commentary refers. Both may be important, if reference to the print version is necessary. The archive layer preserves all information, which may be helpful in some case, and omits all presentation specific features. Unfortunately this is not the last layer. The archive layer is the product of
a transformation of this sort of HTML-Code.

- **migration layer (HTML)**
  202, 6 Krebse] Buchhändlerisch: Remittenden.

This step is necessary, as the edition is originally produced with MS-Word, a not at all desirable choice as an editor's tool, but just legacy.

- **source text (Word**)
  202,6 Krebse] Buchhändlerisch: Remittenden.

So finally we have come to the starting point. This Word version is converted by a commercial tool to the migration layer in HTML., simply because this is to handle more easily and consistently. The next two steps are obviously done by some sort of program. The main aim of the transformation from the migration to the archive layer is to strip off unnecessary graphical mark up and to transform layout mark up which transports meaning into explicit XML-Tagging. This is a task which has to be done just once. Not so the next step from archive to presentation layer. Different users or situations may demand different presentations. It would therefore be nice, if this transition could be done very easily. On the other hand, it is not quite trivial, as also in our simple example we have to sort out not needed tags, to separate and number the notes and to care for proper linking. In other words: a perfect task for declarative programming.

Declarative programming for critical editions

So it is not astonishing that there already exists a mainstream solution with declarative character: XSL, or more specifically XSLT (Extended Style sheet Language for Transformation). XSL was meant as an advanced sort of CSS, but after some work the developers realized that they had created a tool for the transformation of mark up. The real formatting tool was delivered afterwards as XSL-FO. The way XSLT works resembles remarkably the way of

PROLOG. The transformation is done by traversal of a tree, testing the matching of nodes against patterns in the style sheet.

Why not do it with logic programming? Richard O'Keefe has formulated the answer in this manner:

> "My own experience is that having PROLOG, Scheme and Haskell available it'll take a gun pointed at my head or an extremely large bribe to make me use XSLT for anything."

Of course the mainstream character and the extent of support are arguments in favour of XSLT, but there are also some drawbacks. Firstly XSLT is not a universal programming language, which means that not all problems are solvable within this framework. Secondly XSLT is not designed from ground up as a declarative language. In simple examples this is not easily visible, but as applications get more complex you see the barely disguised imperative background shining through in control structures (xsl:if, xsl:for-each etc.) and pattern matching. A comparison of XSLT and PROLOG is to be found on the SWI-Pages **[**4**]**. There is also some repulsion of XSLT in the literature (cf. [3]).[1]

Anyway, it makes sense to give PROLOG a try, and the first success is to see how seamlessly a XML-document converts into a genuine PROLOG structure. Essentially each XML (also HTML)-tag can be represented in a triple of tagname, attribute list and content (possibly also a list or empty). The rest is nesting. Meanwhile there are some libraries available to support this sort of transformation in both directions. As far as I know the first was the initally for HTML designed Pillow from M. Hermenegildo (TH Madrid) [2], which does also some cgi- and http-support and which was used in our project. A hypertext reference takes the form of

a([href='link address'],"link text"),

where the attributelist can be omitted if empty,
which is equivalent to and internally represented as

env(a, [href='link address'],"link text").

An alternative with similar functionality but somewhat more theoretically refined is the field notation by D. Seipel (Uni Würzburg).) [5]. A triple is here by means of operator definition represented in the form

tag:attributelist:content,

where attributelist is an associative list of attribute-value-pairs in the form a:v. The PROLOG-library FnPath goes beyond the pure transformation by providing additional means to select and handle substructures, comparable to XML Query or F-logic.
Further to mention is a library xml.pl, which is delivered with the actual version of Sicstus and which provides also a (simple) matching of substructures aside from the conversion of XML to PROLOG terms and vice versa.

---

[1] Partially this critique is directed generally against declarative programming, but my impression is that the authors are only acquainted with the XSLT-style of it.

It is intended to test these alternatives in the future course of the project. Meanwhile these things are not handled so systematically but rather in an ad hoc manner. The basic structure of the transformation is simple: With the help of the Pillow library the xml-code is transformed to PROLOG-Terms. A few lines of code suffice to traverse the document tree, essentially and within the frame of our purposes the function of an XSLT-processor. The equivalent of the style sheet itself are predicates of the form

umf(Path,env(TagOld,AttLOld,ContOld),env(TagNew,AttLNew,ContNew))
.

In the case a Term matches the TripleOld, and of course all capacities of the PROLOG unification can be used to do this matching, it will be replaced with the TripleNew. Path stores the tags above the current structure and may be used for further narrowing of the unification. In many cases a fact of this form will be enough, to change, delete or add content or attributes just as you like.

But sometimes it won't do and in these cases the facts can be substituted by whatever complicated complicated rules. Our example of the tiny footnote number is such an example. Aside from doing the numbering and the splitting of the notes the interesting feature is the use of grammar. It cannot be seen from our one example, but the structure of the commentaries has some variation. The lemma may be one word or some or a range of words. Also page and lines may be single numbers or ranges. This type of variance is very well handled by grammars.

```
umf(Pfad,env(notep,AttL,[env(pg,AttL2,Cont)|Rest]),''):-
        tagflag([69,114,108,228,117,116,101,114,117,110,103,101,110]),!,
        retract(note_count(N)),
        O is N + 1,
        assert(note_count(O)),
        number_chars(O,OChs),
        number_atom(O,OA),
        phrase(stellKomm(stelle(PageCh,LineCh,LemmaList),Inhalt),[env(pg,AttL,
        Cont)|Rest],Inhalt),
        atom_chars(Page,PageCh),
        atom_chars(Line,LineCh),
        jpd(First),
        cat(DatN,First,[3,3]),
        DatN=[Jpk,Num,Ext],
        cat(['jp',Num,'n',OA,Ext],NDatN,_),
        cat(['out/',NDatN],PfadDatN,_),
        assert(quer_verw(Page,Line,NDatN)),
        one(vorspann(Vorspann)),
        one(xml2terms(Vorspann,VorTerms)),
        tell(PfadDatN),
        one(output_html([VorTerms,env(note,[id=OA, pg=Page, ln=Line],Inhalt)])),
        told.

stellKomm(stelle(Page,Line,[Lemma1,Lemma2]),Inhalt) -->
```

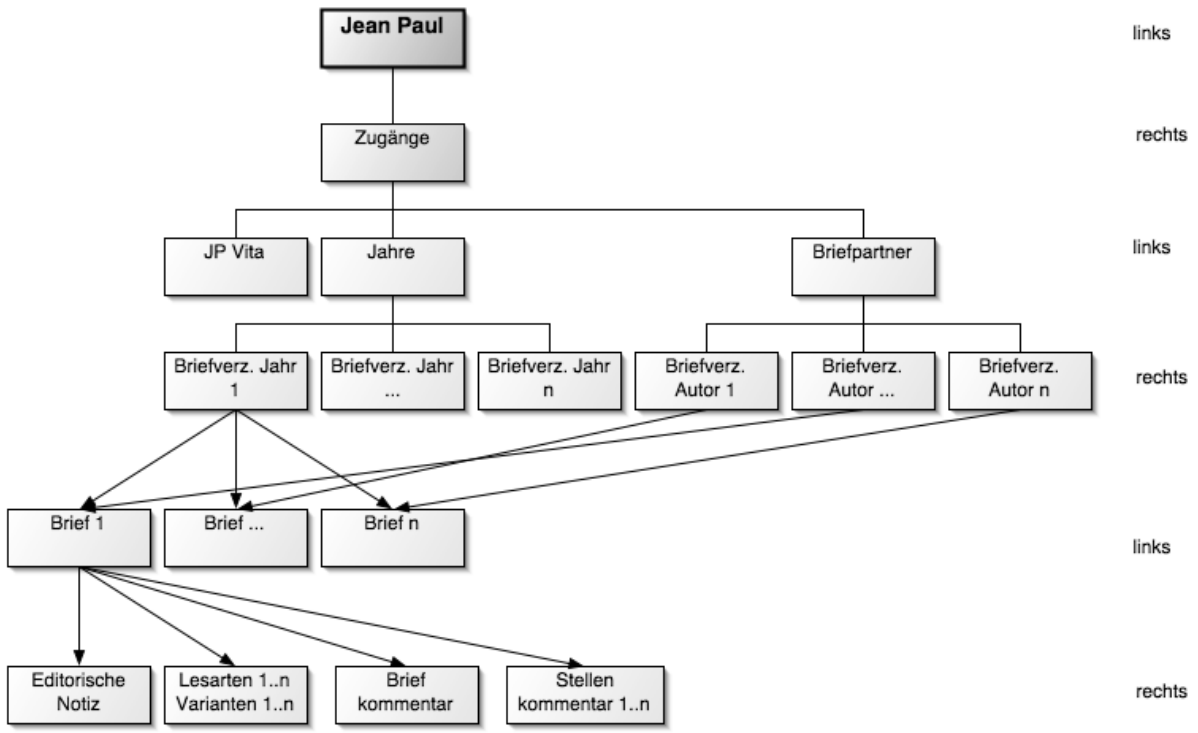
```
        page(Page),line(Line),page(Page2),line(Line2),lemma(Lemma1),
        bis,lemma(Lemma2).
stellKomm(stelle(Page,Line,[Lemma]),Inhalt) -->
        page(Page),line(Line),page(Pag2e),line(Line2),lemma(Lemma).
stellKomm(stelle(Page,Line,[Lemma1,Lemma2]),Inhalt) -->
        page(Page),line(Line),lemma(Lemma1),bis,lemma(Lemma2).
stellKomm(stelle(Page,Line,[Lemma]),Inhalt) -->
        page(Page),line(Line),lemma(Lemma.
page(Page) --> [env(pg,_,[RawPage])],{
        append(Page,[44|_],RawPage)}.
line(Line) --> [Line].
lemma(Lemma) --> [env(jpzit,_,[Lemma])].
bis --> [[32,98,105,115,32]].
bis --> [[98,105,115,32]].
rest([I1,I2]) --> cont(I1),rest(I2).
rest(env(jpzit,[],Inhalt)) --> env(jpzit,[],Inhalt).
cont(Inhalt) --> [Inhalt].
```

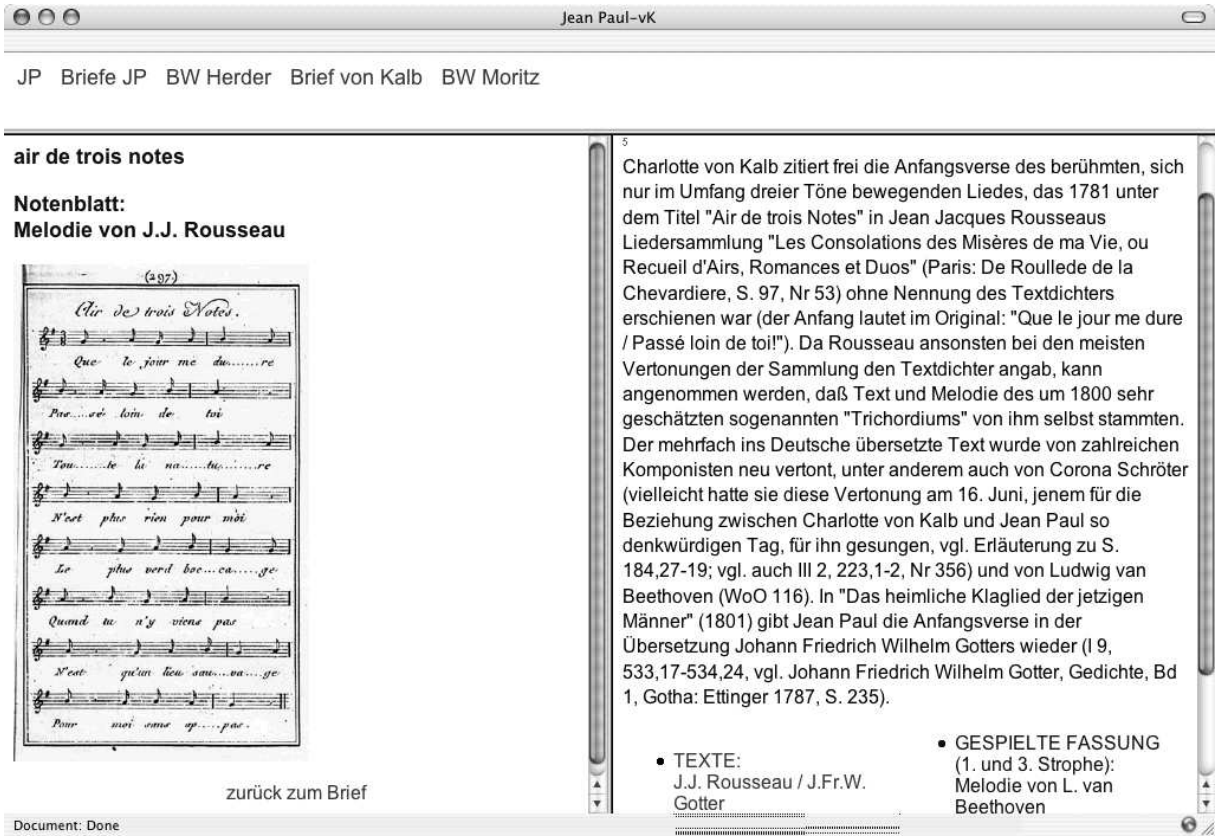
Those are the moments, I prefer to have available the full capabilities of a universal declarative language like PROLOG – and I assume they are the same O'Keefe thought of, when he spoke of the gun pointed to his head.

Topology and navigation

It was already indicated above, that logic programming is used for two different tasks within the generation of the critical edition: the transformation of the original text into the archive layer format and then to convert this into the presentation layer. The latter is not to be confounded with the graphical surface. This is to be stressed, as it is opposed to the pure dogma of XML, which distinguishes only a level of content from a level of graphical presentation. I wholeheartedly agree that this distinction is to be made. The ideological part is the opinion that there exists one level of content independent of different users and user interests. The presentation layer is not meant as the graphical surface, which is primarily produced by the style sheet and the browser, but as an organisation of the content with regard to different user interests. Not the only but a very important task in this context is navigation. I suggest to distinguish between topology and navigation with reference to a document. Whereas topology refers to the potential connections of document nodes, navigation means the realised and used connection paths. You can use public transport as a metaphor, where the streets or waterways would provide the topology, whereas the de facto offered connections correspond to the navigation.
In accordance with this convention, the providing of the topology would be part of the archive layer whereas the presentation has to care for the navigation. That means also while topology being a matter of conceptual and structural relations, navigation has also to take into account social aspects of user interaction as well as technical and aesthetic aspects of the realisation.

If we focus on the letters to Jean Paul, a navigation with the main access over years or correspondence partners could be designed like this





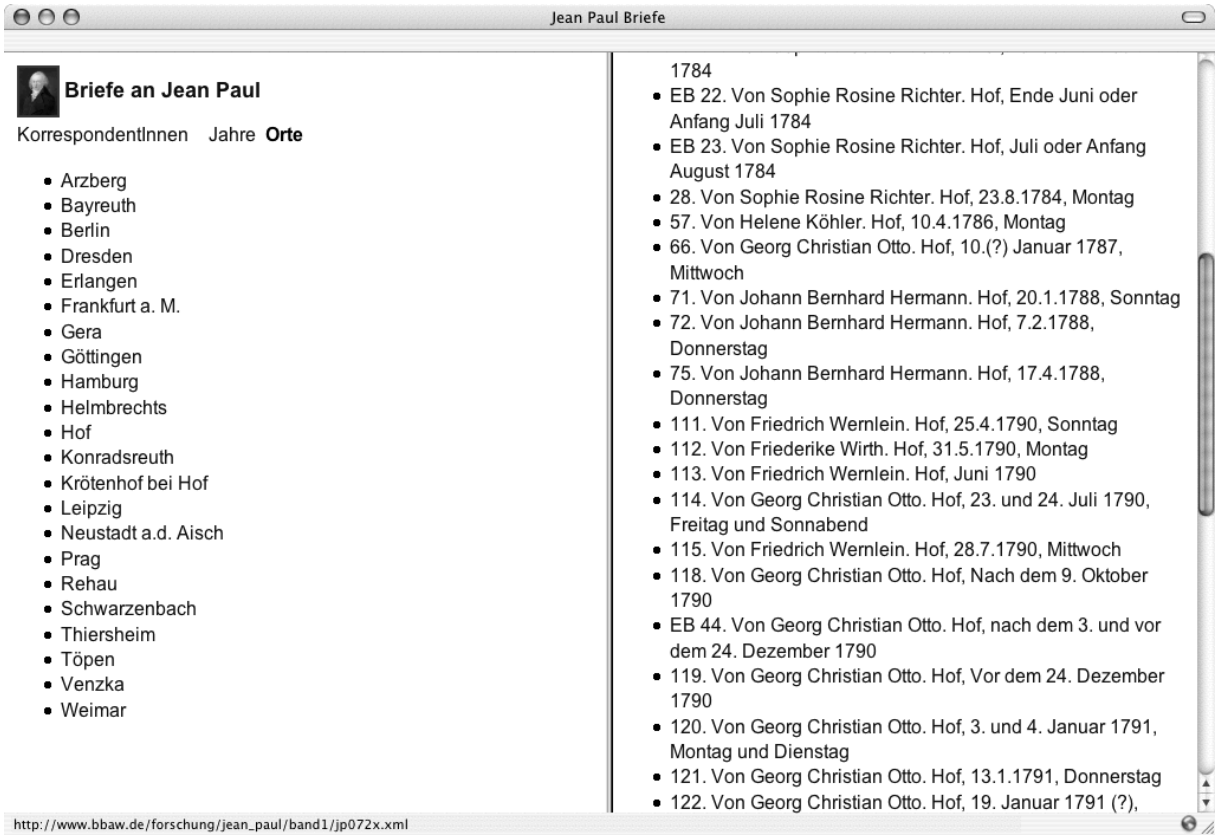
Obviously we could conceive a lot of other user interests and corresponding ways of access: e.g.

- a portrayal of person and work of Jean Paul
- literary correspondences about 1800
- poets in the times of classicism and romanticism.

We might meet new contents in these examples, e.g. biographies or illustrative material, but primarily we need new and specific ways of access, which would mean for example a navigation not centred around one person, when giving a picture of the correspondences or the poets of the time. But all of these navigations could be produced from the one archive layer, or better: the archive layer should be designed in a way that multiple navigations can be produced with little effort.[2]

The information from the document, even in its tagged archive format, is not enough to produce the navigation. We need meta information, in our example at least on the place, date and author of the letters. This information is kept in a relational table and here again it is very comfortable that PROLOG handles these as an own genuine structure. Derived from these are as well the headlines of the letters as the different directories of persons, years and places, which provide access to the content. You may see the presentation layer as a virtual layer on top of documents and meta information.

[2] Navigation is not the only thing, the presentation layer is responsible for. Part of his tasks is also providing different formats and media of output.



Nonetheless the handling of meta data is at the moment a rather modest matter in our project. Next step will be to incorporate meta information in RDF and so to get nearer to the semantic Web. Without doubt this is a very interesting and important development for the future of critical editions, even if we don't know what exactly the impact will look like. But one thing is sure: As RDF is a format which must be supplemented by a separate means for inference,[3] PROLOG (and companions) will be our favourite also in this field.

Literature

[1] Dan Brickley, Enabling Inference (http://www.mozilla.org/rdf/doc/inference.html)

[3] Examples of logic programming tools and libraries for RDF are to be found for example with SWI-PROLOG and with the Mozilla-project [1]

[2] Daniel Cabeza, Manuel Hermenegildo, WWW Programming using Computational Logic Systems

[3] Michael Leventhal, XSL Considered Harmful, www.xml.com/pub/a/1999/05/xsl/xslconsidered_1.html

[4] Bijan Parsia, Long story about using SWI-PROLOG RDF and HTML infrastructure, esp. Ch. 6: DCGs Compared to XSLT (http://www.xml.com/pub/a/2001/07/25/prologrdf.html)

[5] Dietmar Seipel, Processing XML-Documents in PROLOG

[will be completed]