Dictionary Encoding Based on CSS and XML/HTML Parsers

1. Source File Structure

The source file has been obtained as a result of pdf-to-doc-to-odt-to-xhtml conversion. Resulting xhtml file preserved almost all typographic features except correct column rendering. The reason of the conversion is to perform annotation on the text's source level using OHCO model, not as we did previously in WYSIWYG editors. CSS and XHTML parsing is based on the following tag hierarchy:

Listing 1. Object hierarchy of the source file

```
<html>
        <head>
        <! - - … - - >
        <body>
                <div>
                        <p>+
                                <span>text</>+
                        <div>+
                                <p>+
                                        <span>text</>+
```

2. Parser Output Structure

Listing 3. Parser Output's General Schema in RELAX NG

1. dict_p = element dict{(entryGroup_p | entry_p)+}
2. entryGroup_p = element entryGroup{entry_p+}
3. entry_p = element entry{form_p, senseList_p, subEntryList_p?}
4. form_p = element form{attribute parent{"entry"|"sub-entry"}, attribute type{"simple"|"variant"|"inflected"|"compound"}, orth_p, form_p*}
5. orth_p = element orth{text}
6. senseList_p = element senseList{sense_p+}
7. sense_p = element sense{translation_p+, subEntry?}
8. translation_p =element translation{text}
9. subEntry_p = element subEntry{form_p, senseList_p}

3. Interpretation of Text Features

Lexicographic structure of the dictionary is presented by means of typography (font features, layout) and syntax (predefined indicators and punctuation). Here below we provide some interpretation rules, which we used for parsing, in [text feature] : [interpretation] format:

Listing 2. Dictionary interpretation rules

1. Font Features
   1.1. [bold] AND [NOT[enumeration, reference]]: [form element]

1.2. [non-bold] AND [non-italic]: [sense element]
1.3. [non-bold italic] AND [Latin Encoding]: [international names of flora and fauna]

2. Indicators/Punctuation
   2.1. [‘1.’, ‘2.’ etc.]: [entry sense number]
   2.2. [‘1)’, ‘2)’ etc.]: [sub-entry sense number]
   2.3. [Rome Digit]: [homograph is to the left]
   2.4. [‘:’]: [delimits <orth> element from it’s compound form]
   2.5. [‘~’]: [headword placeholder]=[<oRef/> in TEI]
   2.6. [‘, -’]: [delimits <orth> element from it’s inflected form]
   2.7. [’,’ between bold words]: [delimits <orth> element from it’s variant form]
   2.8. [non-latin text between parenthesis]:[definition of sense] OR [context] OR [directing information] etc.

4. Parsing Workflow

At first some structural tokens are defined. After that key tokens are used to identify desirable elements or their boundaries.

Listing 3. The workflow steps and main principle

1. Retrieve lexical data from the source file:
   Defined tokens: [second level div elements (see Listing 1.)]
   Key tokens: [div elements with the largest content]=[page body elements]
   Instruction: [Retrieve all key tokens content]=[lexical data]
2. Markup dictionary entries
   Defined tokens: [all sequences of bold letters along with punctuation and white spaces]
   Key tokens: [those lacking ‘~’ character]=[only entry form elements(not sub-entry)]
   Instruction1: [Mark up all key tokens with <form type = “entry”/> tag]
   Instruction2: [Delimit and markup all entries using entry form elements as a delimiter]
3. Markup form and sense elements
   Comment: [form element is already marked up]
   Instruction1: [Mark up all the content to the right of form element with <sense/> tag]
4. Markup <orth> element and text to the right of it
   Instruction1: [Select first word and mark it up with <orth> tag]
   Instruction2: [Depending on the punctuation character after <orth> element markup text to the right as <form/> having @type = “variant”|”inflected”|”compound”]
5. Markup <senseList/> and <subEntryList/> elements
   Defined tokens: [tokens obtained as a result of full stop based delimitation, i.e. sentece-like constructions; enumeration indicators are ignored]
   Key token: [Token containing greatest enumeration indicator; if no such then first one]
   Instruction: [Using end of the key token as a delimiter markup left part as <senseList> and right part as <subEntryList> elements]
6. Markup sense items inside <senseList>
   Instruction: [Using tokens defined in step 5. markup each one as <sense> and assign enumeration number to the *@n* attribute]
7. Markup and move sense examples, if so, into <sense> constructions
   Tokens defined: [tokens defined in step5]
   Key tokens: [tokens containing <form> tags marked up in step2]

8. Markup all sub-entries inside <subEntryList>
   Comment: [similar principle as used for main entries is followed, see steps:1-4]
9. After general schema is achieved indicated(<lbl>, <usg>, <def> etc.) constructions are marked up and moved, if needed, to appropriate elements.

5. Structure According to TEI P5

Listing 4. The main schema of the dictionary encoded according to XML/TEI, wihout <xr>, <usg>, <lbl> and <def> elements.

1. dict_p = element dict{(superEntry_p | entry_p)+}
2. superEntry_p = element superEntry{entry_p+}
3. entry_p = element entry{form_p, sense_p, re_p*}
4. form_p = element form{attribute type{“simple”}, orth_p, element form{attribute type {”compound”|”inflected”|”variant”}, orth_p}?}
5. orth_p = element orth{text}
6. sense_p = element sense{cit_1+}
7. cit_1 = element cit{attribute type{“translation”}, attribute xml:lang{“Ky”}, quote_p, cit_lev_2}
8. cit_2 = elements cit{attribute type{“example”}quote_p, cit_p_3}
9. cit_3 = element cit{attribute type{“translation”}, attribute xml:lang{“Ky”}, quote_p}
10. quote_p = element quote{text}
11. re_p = element re{form_p, sense_p+}