A Virtualization-Based Retrieval and Update API for XML-Encoded Corpora

Cyril Briquet (1) (2), Pascale Renders (2) (3), Etienne Petitjean (2)

(1) McMaster U, ON, Canada (2) CNRS, Nancy, France (3) U of Liège, Belgium
Take-home message

- context: FEW, ref. dictionary in French & Romance Linguistics

- objective: semantic tagging of a very very complex dictionary

- our desire: offer support for *natural linguistic reasoning*
  = tag-aware text retrieval, tag-aware markup update

- our proposed mechanism (made available as an API):
  virtualizing sections of the XML document as needed

- `<disclaimer>we're not XML experts</disclaimer> <!-- ;) -->`
This afternoon's agenda

- FEW dictionary
- the retroconversion problem
- virtualizing the XML document (concept, API)
- in practice
Französisches Etymologisches Wörterbuch

- reference dictionary in French & Romance Linguistics
- Walther von Wartburg et al., 1922-2002
- historical & etymological
## Shallow comparison: OED & FEW

<table>
<thead>
<tr>
<th>Feature</th>
<th>OED</th>
<th>FEW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages</td>
<td>21730</td>
<td>16865</td>
</tr>
<tr>
<td>Volumes</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>Entries</td>
<td>300 000</td>
<td>20 000 (*)</td>
</tr>
<tr>
<td>Lexemes</td>
<td>600 000</td>
<td>900 000 (est.)</td>
</tr>
</tbody>
</table>

(*) FEW entries are etymons, not lexemes, thus fewer
FEW is very very complex

hard to read:

- complex structure
- large number of fields
- implicitness (syntactic + semantic)

hard to search:

- can't do transversal search in paper version
Retroconversion of the FEW

<< starting from the paper version, how can the complex dictionary structure be automatically extracted into a searchable database? >>

* ongoing project at ATILF lab in Nancy, France
* team of Prof. Eva Buchi, Research Director
* backed by CNRS and Nancy University
The bottom line: an example

<b>completus</b> vollständig; <lb> vollkommen. <lb>  
<p>I. 1. a. Vollständig. — Mfr. nfr. <i>complet</i> „à<lb> quoi il ne manque aucune des parties nécessaires“<lb> (seit ca. 1300, Monstr; Rhlitt 6, 464), [...] saint. St-<lb> Seurin <i>compiet</i>, Minot <i>conpiet</i>, npr. <i>coumplèt</i>. —<lb> Übertragen. Nfr. <i>complet</i> „(pop.) tout à fait ivre“<lb> (seit Flick 1802).

<entry><b>completus</b></entry> vollständig; <lb> vollkommen.  
<doc><p><pnum id="I 1 a">I. 1. a.</pnum> <title>Vollständig.</title> —  
<form><i>complet</i> „à<lb> quoi il ne manque aucune des parties nécessaires“<lb> </form>  
<precisions>(<attestation>seit <date>ca. 1300</date>,  
<biblio>Monstr</biblio></attestation>;<attestation><biblio>Rhlitt 6, 464</biblio></attestation>)</precisions></unit>, [...]
Text-oriented XML documents

FEW article

= text-oriented XML document, complying with XML Schema
(currently not TEI but long term it'll try & align with TEI)

= list of text chunks with interspersed tags
(element hierarchy useless, thus not used)
In-memory data structure

- list of nodes: XML tags or text chunks
- constructed using a validating SAX parser
- UTF-8, entities resolved, character legality enforced
- text normalized (redundant spacing, break tags)
FEW retroconversion workflow
What's in a tagging algorithm?

- detection of dictionary fields
  - text retrieval, markup retrieval
  - keyword search (*dictionary-matching problem*)
  - regexp
- secondary contextual lookups often necessary,
  e.g. find keywords within 10 words of tags containing keyword,
  *in text-oriented representation*
  
- tagging of detected fields (markup update)

- sometimes, modification of dictionary text (text update)
Retrieval challenges

- false negatives:
  tag interference (e.g. exponent, end of line)
  prevents matching of keywords, regexp

- false positives in irrelevant contexts:
  keyword search not relevant everywhere
Use case: preventing false negatives

- Emprunt de lttard. mlt. augmentator (4e–6e s., ThesLL;

in this use case: 4e–6e s. is a datation; full-text query not discarding tags would result in false negative, as none of the 6 fragments (4, e, -, 6, e, s.) alone is a datation

in this use case: <e> tags should be skipped

Emprunt de lttard. mlt. augmentator (4e–6e s., ThesLL;
Use case: preventing false positives


- in this use case: 1750 is a date, 1787 is not; full-text query only discarding all tags would result in false positive

- in this use case: <biblio> elements should be made invisible

Nfr. complètement „action de mettre au complet“ (seit 1750, text in )
Update challenges

- updates may be far from matches, i.e. in non-collateral branch of tree representation
- updates may span several text chunks, with interferences from legitimate tags in-between
- match points required to offer support for *natural linguistic reasoning*
Virtual string

- **Definition:** concatenation of adjacent text chunks, except those within elements configured to be invisible

- sections of XML document virtualized into multiple virtual strings separated by visible tags

- backed by underlying XML document; updates are transparently propagated
Text virtualization example

visibility: V visible, I invisible, S skipped, T terminal

3 virtual strings, tag last 2 words of middle virtual string:

• … `<V>some nice text</V> <I>and text to be made invisible</I> and now <S>finally</S> <V>nice text again</V></T> ... 

• … `<V>some nice text</V> <I>and text to be made invisible</I> now `<NEW>now <S>finally</S>` `</NEW>` `<V>nice text again</V></T> ...
API overview

virtual text retrieval API, virtual text update API, virtual tag splicing API

some nice text virtual string

text virtualization API

<X><Y>some nice text</Y> <Z>and text to be made invisible</Z> and <W>finally</W> ...

addressing API

text retrieval API, text update API

virtual string and finally

text virtualization API

read this slide bottom-up, please :-)
VirtualTextSearcher searcher = new VirtualTextSearcher(iterator, partition);
for (VirtualString vs : searcher) { // text virtualization
    Set<KeywordMatch> matches = fewPrefixBase.findAllKeywords(vs.getText());
    VirtualTagSplicer virtualTagSplicer = createVirtualTagSplicer(this,vs);
    for (KeywordMatch m : matches) {
        int startIndex = ...; int endIndex = ...;  // virtual text retrieval:
        if (isLicitPrefix(vs,endIndex) == false) continue; // requires match point
        endIndex = getExtendedPrefixKeywordEndIndex(vs,endIndex);
        virtualTagSplicer.markSubstringForTagging(startIndex,endIndex,affix,
                       new String[] { "type", "descendance" },new String[] { "prefix", "etymon" });
    }
    virtualTagSplicer.spliceAll(); // virtual tag splicing
}
Natural linguistic reasoning

- retroconversion of FEW = \textbf{breakthrough}
  - familiar level of abstraction: text without tags
  - flexible specification of retrieval & updates

- similar projects
  - abstraction level too far from dict.: tags everywhere
  - hard to specify: long regexp containing tags
In practice

- Java implementation: 64kloc (API core: 7.5kloc)

- 144 articles retroconverted (~0.75% of FEW)

- coverage: 98.5% automatically tagged

- precision and recall of tagging:
  - depend on accuracy of linguistic analysis, not on API (which returns exact results)
  - difficult to measure, takes days to tag manually
What about XQuery?

- XQuery Full Text extension:
  FTIgnore option configures tag visibility during search

- XQuery Update Facility

- returned results = XML elements... 
  not text with support for match points...
  but at this point the tagging algorithm is just getting started
  => how to perform additional contextual search & updates?
  (we just don't know...)
Next steps

- package API into dedicated library
- get feedback on syntax, semantics
  (to what extent does the API overlap with and/or benefit from and/or contribute to existing related technology?)
- optimizing current implementation for
  - speed: addressing, virtual text upd., virtual splicing
  - memory usage: text virtualization
Take-home message

- context: FEW, ref. dictionary in French & Romance Linguistics

- objective: semantic tagging of a very very complex dictionary

- our desire: offer support for *natural linguistic reasoning*
  = tag-aware text retrieval, tag-aware markup update

- our proposed mechanism (made available as an API):
  virtualizing sections of the XML document as needed

- `<disclaimer>we're not XML experts</disclaimer> <!-- ;) -->`
Thank you