Building a Dictionary Using XML Technology

Alberto Simões[1], José João Almeida[2], and Ana Salgado[3]

1 **Centro de Estudos Humanísticos, Universidade do Minho, Braga, Portugal**
ambs@ilch.uminho.pt
2 **Centro Algoritmi, Universidade do Minho, Braga, Portugal**
jj@di.uminho.pt
3 **Instituto de Lexicologia e Lexicografia da Língua Portuguesa, Academia das Ciências de Lisboa, Lisbon, Portugal**
anacastrosalgado@gmail.com

Abstract

In this article we describe the workflow implemented to convert a dictionary saved as a PDF file into an XML document and posterior importation into an XML aware database, and the process to edit, add and delete new entries. The conversion process was challenging given the format of the PDF file, and the fine grained detail of the XML schema that was used. For that, an iterative filtering approach was used. To store the dictionary we decided to use an XML aware database (eXist-DB), that stores each dictionary entry as a separate resource. It can be queried used a web interface developed using XQuery. The lexicographers can edit entries using the oXygen XML editor, reading and storing them directly in the database. In order to guarantee incremental backups, it was defined a mechanism to import the XML database into a GIT repository. Finally, a couple of programs were created in order to prepare regular reports on the dictionary revision process, as well as to backup it in a GIT repository.

1998 ACM Subject Classification I.7.2 Document Preparation / Markup languages

Keywords and phrases XML databases, dictionaries, XQuery, PDF files

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.14

1 Introduction

After the release of the *Dicionário da Academia das Ciências de Lisboa* (DACL) in 2001 [1], our goal is to recover that work, update the dictionary data and publish it both on the Internet, as a web application, and as a conventional paper dictionary.

We do not intend to publish the dictionary, or even make it available to the public, as it is. Our aim is to manually revise the full dictionary, fixing known errors, detecting others, and including new terms. That is, we want to create a new version of the dictionary to be made available in the web for free by the end of the next year.

The main problem arose when it was found that the only source for the dictionary itself, was from a PDF file. [1]. There was no time nor money to allow the full transcription of the document. This required an automated process to recover the data from the PDF file. In order to achieved this, our previous recovering other dictionaries [3, 5] was crucial, and allowed this process to be faster.

[1] This PDF file was created from a Word document, that was generated from a Microsoft Access file, but none of these files were still available.

This process resulted in a text file with font face and size information. Then, a set of rewriting rules were written, to convert this information into a basic XML structure that was later enriched using an iterative filtering approach, as we will describe.

The resulting XML was split into smaller documents, one for each dictionary entry. These documents were later imported into an XML aware database. In this case, we chose eXist-DB [4].

In order to allow the revision and enrichment of the dictionary, it was needed an interface or some other mechanism to allow linguists to edit the dictionary records. With that in mind, we created an application based on XQuery to allow the navigation of the dictionary, and rendering of links using the XMLDB protocol, that oXygen XML editor[2] could understand, making it possible to edit an entry using a single mouse click.

Finally a set of reporting and export tools were developed in order to monitor the pace of the dictionary revision, and the export of the dictionary into different formats.

In the next section we will focus on the task of understanding the PDF document format and exporting it into XML TEI [7]. Follows Section 3 that explains how the dictionary was imported and indexed into a XML-aware database. Section 4 present the developed tools to help maintaining and validating the dictionary. Finally, we conclude in Section 5 with some insights on next steps on the development of the new dictionary.

2 Rewriting PDF into XML

In previous projects we had already applied techniques in order to convert dictionaries encoded in different formats into some XML schema. Each one of these data conversion tasks resulted in very different challenges. The same holds true for the PDF format conversion of the *Dicionário da Academia das Ciências de Lisboa*.

As described earlier, the PDF file was generated from a Microsoft Word document. The format is a two column template, using mostly text in a same font-size, but changing its style, including normal, italic, bold and small capitals. The tools we tried to convert the PDF to text did not provide satisfactory results:

- *pdftotext* command line utility loses all formatting, which means that all the information codified as different font styles is completely lost.
- *pdftohtml* as similar problems, including the fact that it also lost all non-ASCII characters, namely the phonetic transcriptions (using modified IPA) and the etymological information which could include Greek words. Note that *pdftohtml* includes an option to output XML, but it is unable to detect some user-defined fonts. Although not tested, we expect *pdf2xml*[3] to behave similarly, given it uses, just like *pdftohtml*, the *Xpdf* library. It was not tested as its source code is not prepared for easy installation, lacking the usual configuring mechanism present on most OpenSource tools[4].
- OCR tools, like the free *Tesseract* OCR, and commercial tools, like *Omnipage Pro* and *Abbyy Fine Reader* converted the vectorial PDF files into images, and then tried to recognize the text. Their results were also far from satisfactory even when using high quality images. Also, during this process, when specifying Portuguese as the main language, they would miss the detection of non Portuguese words.

[2] `https://www.oxygenxml.com/`
[3] `https://github.com/eliask/pdf2xml`
[4] There are some binaries for the *pdf2xml* tool in SourceForge, but it is known that SourceForge was hacked more than once, and therefore binaries available there are not to be trusted.

```
<text font="KIKNHC+Garamond-Redondo"  size="9.548">v</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">o</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">g</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">a</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">l</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548"> </text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">c</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">e</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">n</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">t</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">r</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">a</text>
<text font="KIKNHC+Garamond-Redondo"  size="9.548">l</text>
```

Figure 1 XML as generated by PDFBox to the "*vogal central*" string.

Apache PDFBox tools[5], on other hand, proved to be very effective.. This tool is able to process a PDF file and generate a (large) XML file, where each character in the PDF file is encoded into an entity with the typeface and font size, as shown in figure 1. This XML file takes 2.7GB of disk space, with 28.5 million `text` tags.

This XML was then simplified, joining all text with he same typeface and font size into single elements. Then, a list of different pairs $(font, size)$ was created, and mapped to different tag elements. For the common typeface and font size adding a bold, italic or small caps were used the elements *b*, *i* or *sc*. For other, less clear strings, other *ad-hoc* elements were defined.

A set of rewrite rules were then written in order to process this file and identify what role each string played in the dictionary, determine what was the entry term, its phonetic transcription, sense identifiers, synonyms, quotes, etc. The techniques employed here were similar to the ones applied previously [5].

After this step, our main challenge was the granularity of the desired markup, which was more fine-grained than in previous works which made the XML processing time step take too much time. A simple addition of a rewrite rule could make the entire process to take up to 30 minutes. This kind of approach was hard to maintain, as it was not easy to perform simple tests. Also,

At some point we noticed that some entries were already correctly formatted, and we were concerned that writing rules for more complex ones could affect the correct entries. At this point, we implemented an iterative filtering mechanism. First, the XML file was divided into multiple files, one for each entry. Then, the following algorithm was applied:

1. define a basic DTD for simple entries;
2. test all single entry files against the DTD and move the valid ones into another folder;
3. include a set of new rules to process the remaining files in order to annotate extra information;
4. update the DTD to support the new elements and structure, while maintaining the validity of previous validated files;
5. go to the step 2.

[5] `https://pdfbox.apache.org/`

This approach was quite useful, not only because we guaranteed that new rules would not damage entries already correctly formatted, but also allowed us to define the minimum needed DTD: elements, their arity and attributes were added only when there was an entry needing for them.

In the end, a list of about two hundred entries were not validated by the DTD. At this point we noticed that most of the errors were related to mistakes (incoherences, mostly) of the original edition. As the original dictionary was created based on a Microsoft Access database[6] without integrity constraints on the entry contents, it was not possible to guarantee the quality of the work. Due to this, mistakes that were found after producing the dictionary Word document were correct manually on the document itself. Also, some characters were added or removed in order to guarantee some graphical format but, at the same time, break the consistency with the dictionary entry format that was enforced by the Word document generation process. These entries were edited manually using a tool developed specifically for this purpose. This tool allowed the user to edit an entry and save it after being validated.

This iterative process made it possible for all 69,428 entries of this dictionary to be validated against the created DTD.

3 Indexing the XML dictionary

The obtained dictionary has some errors, such as words broken in two, given the transliteration not being detected by our previous tools, or some of the phonetic and Greek letters not being detected. Unfortunately most of these errors will need to be fixed manually.

If we had only one lexicographer working on this dictionary, it could be possible to use a single file, and allow to edit it at once in an XML aware editor. With the intent of allowing more than one person to edit the dictionary, and given that the complete dictionary is, formally, a sequence of the same element (*entry*), we split the file into 69,428 smaller XML files. This also allows the editor to open and validate each entry much faster, than analyzing the full file. In order to allow the better understanding of the discussion that follows, listing 1 presents the content of the dictionary entry for the terms *vassoura/vassoira*.

A dictionary entry can have more than one headword. As a simple example, the term "*vassoura*" (broom) can also be written as "*vassoira*". So, it is not possible to rename each one of the XML files to the term it defines (at least, not for every entry, or we would have duplicated entries, saved with different names). To suppress this problem, we decided to use the first orthographic form of the entry. In case of polysemous words (words appearing as the first orthographic form in more than one entry) we concatenated the word name with the acception number (separated by an underscore).

This process makes it easier to find an entry. But for some situations it is still difficult. How will the lexicographer know if a specific word is the first orthographic form of its entry, or even, if a specific word has more than one meaning registered in the dictionary? Therefore, we needed to develop a mechanism to allow the lexicographer to search for specific terms, and looking to their entries' contents, choose the one to be edited.

Instead of developing an in-house solution for this problem, we decided to give a try with an XML-aware database. The choice was the well known eXist-DB [4], not just because it is free and open-source, but also because it has extensive documentation.

Our long term plan is to develop a simple interface to allow the maintenance of the dictionary, based on the XForms [2] standard. Meanwhile, while that is not possible, and in order to allow lexicographers to start their work revising the dictionary, we adopted oXygen

[6] Unfortunately this database is lost, explaining why all the work on re-engineering the PDF document.

Listing 1 Entry for the word *vassoura/vassoira* encoded in XML.

```
<entry [...]>
  <term>
    <orth>vassoura</orth> <orth>vassoira</orth>
    <pron>vɐs'oɾɐ</pron>  <pron>vɐs'ojɾɐ</pron>
  </term>
  <gramGrp>s. f.</gramGrp>
  <etym>Do lat. <mentioned>*versoria</mentioned>, de <mentioned>versus
</mentioned>, part. pas. de <mentioned>verrĕre</mentioned> 'varrer'</etym>
  <sense n="1">
    <def> Utensílio doméstico formado por um cabo longo ou curto ao qual é
fixado, numa das extremidades, um feixe de folhas de palma, piaçaba,
sorgo, pêlos naturais ou artificiais... e que serve para varrer o lixo.
<quote type="example">O cabo da vassoura partiu-se. Deitou fora a vassoura
porque tinha os pêlos gastos.</quote></def>
[...]
  <sense n="7"><usg type="geo">Bras.</usg>
    <def>Pessoa que ganha sempre ou quase sempre em sorteios, jogos de
azar... </def></sense>
[...]
</entry>
```

XML Developer[7]. This choice was backed by the tight cooperation between the developers of eXist-DB and oXygen. Example of that cooperation is the wizard available to connect to eXist-DB from oXygen. With some extra work it was also possible to tweak Mozilla Firefox to open URIs using the `oxygen://` protocol. This allows the lexicographers to do a query in the eXist web pages, search for the entry to edit, and open it with a simple click on a link.

Finally, to make the 69K documents searchable, the eXist-DB collection was configured to be indexed by Lucene[8]. Note that Lucene is part of eXist-DB and does not need to be installed separately. The only requirement is the creation of a configuration file, to enable full text search, and specifying which elements of the XML documents are to be indexed. For the dictionary we created two different indexes with two very distinct goals:

- The first is an index for the entries' orthographic form. These index allows the lexicographers to search for a specific term entry. Given the index is only over the content of an element, it is quite small and efficient.
- The second index allows the reverse-search [6] of the dictionary. This type of search mechanism is quite interesting when analyzing a dictionary, as it allows to search for entries not by their head word, but using their definition. This index is quite large, as it contains all the dictionary text.

4 Developed Tools

When choosing eXist-DB to store our XML documents we ended up choosing a complete solution for web application development. Although the database can be used using different APIs, like REST or xmldb protocols, eXist-DB suggest users to create applications on top of it. eXist-DB allows the development of web applications using standard W3C protocols, like XQuery and XForms. It also includes a full web Integrated Development Editor that allows the programmer to run queries but also to edit the application code.

[7] `http://oxygenxml.com/`
[8] `https://lucene.apache.org/`

Listing 2 XQuery script to validate the dictionary collection.

```
xquery version "1.0";
declare namespace validate="http://exist-db.org/xquery/validation";

<reports> {
    for $doc in collection("/db/academia")
    let $file := fn:base-uri($doc)
    return <file uri="{ $file }"> {
        validate:jaxp-report(doc($file),true())
    } </file>
} </reports>
```

Instead of developing our tools in an external programming language, we decided to test how far we could go with eXist-DB. Not just for the sake of analysis of the tool, but also for portability. It would be much easier to port the dictionary application to the final servers if it uses just one technology.

The next list describes briefly the tools we implemented to help in the management of the dictionary. Note that some are simple XQuery scripts, to be run in the terminal, while other are end-user interfaces, developed for the web.

Validator
The first task was the development of a script to validate every entry in the database. Although eXist-DB supports different kind of validations, we preferred to create a standalone tool. This allows us to remove liberty points in the schema, turning it more restrict, and test how many entries would be affected. As simple as this script may seem, it took some time before it could validate all the collection entries in a reasonable time. Therefore, listing 2 presents our XQuery script. This script takes about 3 minutes to validate the 69K entries in a Quad-core Xeon 2.40GHz, outputting an XML document with existing errors.

Search
As explained before, the lexicographers use a web application to query the dictionary and obtain the filename where the entry is encoded. This script allows both the search by an orthographic form (searching entries in the `orth` element) or doing reverse search (looking up in every PCDATA section of the XML document). The XQuery script returns the complete entries, in the original XML format. A Cascading Style Sheet is then used to make the content adequate to be viewed in a web browser. Figure 2 shows the *vassoura/vassoira* entry as presented currently in the web application[9].

New entry
As stated earlier, at the moment the lexicographers are using oXygen to edit the dictionary. To create from scratch a dictionary entry is not a simple thing, specially when the user XML competences are not strong. To simplify this process, a small XQuery script that, based on the term, validates if it already exists, and in case it does not, create a boilerplate XML document in the database, that can then be edited.

List domains
Although lexicographers will review all dictionary entries, before the dictionary publication, the definitions of some words are prepared by specialists in different areas (like mineralogy or astronomy experts). For those, the entries are exported as rendered in the browser, so

[9] Note that currently the CSS is hiding the phonetic transcription, and that the last part of the entry, with the diminutive form of the word, is being wrongly considered part of the last definition, and therefore, appearing in the wrong position.

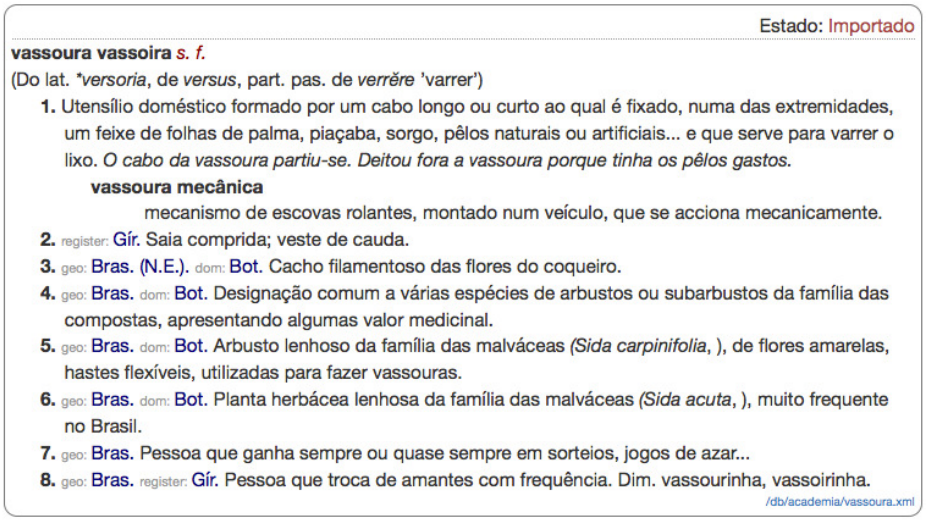

Figure 2 Entry for *vassoura/vassoira* as presented to the lexicographer in the browser.

they can validate them. As most of them are elder and prefer not to use the computer, these lists can be printed. So, a XQuery script was prepared that list all available areas of knowledge, and allow the visualization of all related entries.

Change reports

Another developed tool is the creation of Changes Reports. This XQuery scripts extract the entries that were edited or created in the last week, creating a list of these entries. This script is just a search looking into all documents last write access time.

Backup system

We use a quite original approach for backing up an eXist Database. eXist includes tools to export an entire collection either as a ZIP file, including all the collection documents, or exporting these documents to a folder in the disk (or a complete folder structure). To backup or dictionary we have a regular job, being executed every night, to export the collection to a folder. Then, this folder is committed to a GIT repository. This allows us to have a regular backup, but also a quite incremental system (using less disk space), and easy to replicate (at the moment we are pushing this repository into BitBucket[10]).

5 Conclusions and Future Work

In this article we presented our approach in the process of reverse engineering a dictionary published in PDF, in order to convert it to a fine-grained XML document. We discuss not just the process of reverse engineering (a task that is not new, although it was the first time we did it from a PDF document), but also why and how we store it in an XML aware database.

With the goal of making the dictionary available for editing and validation by different lexicographers, we split the dictionary into various XML documents, one for each dictionary entry. Also, as the process of searching these documents was not easy, a web application

[10] `http://bitbucket.org/`

was developed to search the document collection, and create links that allow the immediate access to each file using the oXygen XML Developer editor.

Having the dictionary being edited by lexicographers, a set of other tools required our attention. For those, we wrote small XQuery scripts that run on top of eXist and allow very different kinds of resources to be built.

Nevertheless, a set of other scripts need to be developed:

- Instead of creating HTML reports of each week work, we intend to create daily and weekly reports of editions, generated as XML documents, imported into another collection. This is a very interesting resource to have, in order to monitor the activity in the dictionary, and having a log on all performed changes.
- A paper dictionary can be born, developed, printed and die. But a dictionary to be available on the Internet needs to be dynamic, allowing the dictionary to evolve following the language and culture. Editing directly the XML file is versatile, but not easy to use. So, we expect to develop a user-friendly editor.
- Currently our web application is restricted to authenticated users. In the future an open interface needs to be available to end-users. Although the simple mechanisms to search for entries are already developed (although restricted), we think there is a couple of other interesting approaches. For example, the synonyms and antonyms annotation can be used to present the dictionary as a graph/WordNet-like structure.
- Although we will make the dictionary available on-line, we still want to be able to create other media, like eBooks or even printed books. For that we expect to create a set of exporting tools.

References

1 João Malaca Casteleiro, editor. *Dicionário da Língua Portuguesa Contemporânea*. Academia das Ciências de Lisboa, Verbo, 2001.

2 Micah Dubinko. *XForms Essentials*. O'Reilly Media, Inc., August 2003.

3 Xavier Gómez Guinovart and Alberto Simões. Retreading Dictionaries for the 21st Century. In José Paulo Leal, Ricardo Rocha, and Alberto Simões, editors, *2nd Symposium on Languages, Applications and Technologies*, volume 29 of *OpenAccess Series in Informatics (OASIcs)*, pages 115–126, Dagstuhl, Germany, 2013. doi:10.4230/OASIcs.SLATE.2013.115.

4 Wolfgang Meier. exist: An open source native xml database. In Akmal B. Chaudhri, Mario Jeckle, Erhard Rahm, and Rainer Unland, editors, *Web, Web-Services, and Database Systems: NODe 2002 Web- and Database-Related Workshops Erfurt, Germany, October 7–10, 2002 Revised Papers*, pages 169–183. Springer, Berlin, Heidelberg, 2003. doi:10.1007/3-540-36560-5_13.

5 Alberto Simões and José João Almeida. Processing XML: a rewriting system approach. In Alberto Simões, Daniela da Cruz, and José Carlos Ramalho, editors, *XATA 2010 – 8ª Conferência Nacional em XML, Aplicações e Tecnologias Associadas*, pages 27–38, 2010.

6 Alberto Simões, Álvaro Iriarte, and José João Almeida. Dicionário-Aberto: Construção semiautomática de uma funcionalidade codificadora. In Alain Lemaréchal, Peter Koch, and Pierre Swiggers, editors, *Actes du XXVIIe Congrès international de linguistique et de philologie romanes*, Nancy, 15-20 july 2013 2014. ALTIF. Section 16 : Projets en cours; ressources et outils nouveaux.

7 Edward Vanhoutte. An Introduction to the TEI and the TEI Consortium. *Literary and Linguistic Computing*, 19(1):9–16, 2004. doi:10.1093/llc/19.1.9.