FoLiA in practice: The infrastructure of a linguistic annotation format

Maarten van Gompel[a] and **Ko van der Sloot**[a] and **Martin Reynaert**[ab] and **Antal van den Bosch**[c]

Abstract

We present an overview of the software and data infrastructure around FoLiA, a **Fo**rmat for **Li**nguistic **A**nnotation developed in the scope of the CLARIN-NL project and other projects. FoLiA aims to provide a single unified file format accommodating a wide variety of linguistic annotation types, preventing the proliferation of using different formats for different annotation types. FoLiA is being developed in a bottom-up and practice-driven fashion. We have invested mainly in the creation of a rich infrastructure of tools that enable developers and end-users to work with the format. This work will present the current state of this infrastructure.

1 Introduction

CLARIN's aim is to deliver an infrastructure for researchers that work with language data and tools. This is impossible without agreeing on standards with regard to data formats. Standardisation is an important prerequisite for good interoperability between the many language tools that have emerged within and outside of the scope of the CLARIN project, and to ensure the various data sets released are usable in practice..

In the field, however, we often encounter an abundance of ad-hoc formats. These are data formats that are often characterised by one or more of the following traits:

- They are only used once, often by one specific tool or for just one specific purpose;
- They are poorly formalised or not formalised at all, i.e. there is a lack of a formal schema and semantics;
- They are poorly documented;
- They are often rigid and hard to extend.

The use of such ad-hoc formats can be considered the opposite of proper standardisation and is to be avoided in any large infrastructure project.

CLARIN adheres to the following principles when it comes to standardisation:

- Open standards are preferred over proprietary standards;
- Formats and protocols should be:
  - well-documented
  - verifiable
  - proven (being used in practice);
- Text-based formats are (where possible) preferred over binary formats.

[a]Centre for Language and Speech Technology, Radboud University
[b]Tilburg Centre for Cognition and Communication, Tilburg University
[c]Centre for Language Studies, Radboud University

At the onset of CLARIN, the Dutch and Flemish Natural Language Processing (NLP) community lacked such a proper standard with respect to linguisitcally annotated text, and ad-hoc formats were prevalent in the field. In the scope of CLARIN-NL project TTNWW, the NWO project DutchSemCor, and the STEVIN project SoNaR, a new format for linguistic annotation was developed as a solution to accommodate their representational needs, and so FoLiA (Format for Linguistic Annotation) was created.

The aim of FoLiA is to provide a practical standard, following a generic paradigm, for the linguistic annotation of primarily written text. To this end, a wide variety of linguistic annotation types is supported.

An extensive overview of the FoLiA format is presented in earlier work (van Gompel and Reynaert, 2013), which also features a comparison with competing standards and full motivation for the creation of FoLiA. Documentation of the format is also available elsewhere (van Gompel, 2014) and offers a reference guide to all elements and attributes that FoLiA defines. A brief summary of key features will be repeated in Section 2, but for the remainder we refer to these two prior works.

In this current paper, we intend to focus on the *practical* nature of the format, or rather, on the infrastructure that is built around the format and the ways in which it had been put to use in the scope of CLARIN and beyond. Section 3 will explain our philosophy behind FoLiA and its infrastructure. Section 4 subsequently describes the currently available software infrastructure for FoLiA. Section 5 describes some corpora that have been delivered in FoLiA.

2 Overview

FoLiA is an XML-based format and defines specific XML elements for *structure annotation* (e.g. paragraphs, sentences, word tokens, lists, figures..) and *linguistic annotation* (e.g. part-of-speech, dependency relations, syntax, named entites, etc..). FoLiA makes use of a combination of inline and stand-off annotation, making proper use of the hierarchical nature of XML and facilitating the job for parsers where possible. The format is fully language and tagset independent as tagsets are defined seperately in *FoLiA Set Definitions* by users and never prescribed by FoLiA itself. Validation can proceed on a shallow level, against a RelaxNG schema, as well as a deep level which validates used tagsets against the set definition files.

The sets are at the core of the FoLiA paradigm, annotation elements take an generic attribute named "class". These *classes* pertain to a set and are defined by whatever set definition the user decides to use. The set definition defines all allowed classes and allows for links with data category registries for formal semantic closure.

Other generic attributes besides "class" are attributes to denote the annotator of a particular annotation, the annotator type (human or machine), the confidence level of the annotation, the time of the annotation, and more.

FoLiA also allows for various types of *higher-order annotation*, such as the ability to include alternative annotations, as well as extensive support for corrections on annotations. There is also the ability to link other modalities such as audio fragments of speech, to structural elements. So even though FoLiA is a primarily a format to annotate text documents, speech transcripts are supported as well.

For metadata CLARIN-NL has committed to the CMDI standard (Broeder et al., 2011). Although FoLiA has simple native support for metadata, we see no sense in reinventing the wheel and FoLiA is ideally used in combination with an external metadata format such as CMDI whenever extensive metadata is desired. A reference to the metadata file can be made in the header of the FoLiA document.

3 Our philosophy

Recall the CLARIN principle that a format should be proven and used in practice. FoLiA has been designed in a bottom-up fashion taking especially this principle to heart. Our focus is to solve real problems people face in the field with regards to their linguistic representation needs, and to do so in a generic fashion. The ambition is to deliver a single unified file format that can effectively handle a multitude of annotation needs in a generic fashion. The main motivation is to prevent the need to switch formats whenever an extra annotation type is introduced, and to prevent the scenario in which a plethora of different formats is used for different annotation types.

It is nevertheless always conceivable that a user's particular need is not yet covered by the latest version of FoLiA; in such cases we gladly hear from the user and expand FoLiA where necessary, in collaboration with the user. The development of FoLiA has already proceeded for several years in such a collaborative fashion, and various annotation types have been added in close contact with end-users both from within CLARIN and beyond.

In our philosophy, the creation of a file format is useless without similtaneously creating an infrastructure of tools to work with said format. This has therefore been our main focus over the years and will the the subject of the next section.

4 Software Infrastructure

When we speak of a FoLiA software infrastructure we refer to a published set of software, from whatever sources and for whatever architecture, that enable people to work with FoLiA. Such an infrastructure in simple terms encompasses anything that can either process or deliver the data in the format, we can subdivide it into the following components:

1. programming libraries;
2. tools for validation;
3. tools for conversions from and to other formats;
4. tools for visualisation;
5. tools for searching/querying;
6. editing tools;
7. special-purpose tools; i.e. specialised tools that use the format but are not necessarily focussed on it. In the case of FoLiA, This includes Natural Language Processing or Information Retrieval tools that use the format as a input and/or output.

The programming libraries and tools that are purely designed to visualise, manipulate, or convert the format in basic ways can consider part of a *core layer* of the infrastructure, whereas the special-purpose tools can be considered to constitute an outer layer.

As FoLiA is an XML-based format, the rich and well-established XML infrastructure is open to its users as well. In fact, almost all FoLiA tools effectively rely on the existing software infrastructure available for XML.

It is possible to not use any of the FoLiA-specific tools and use the infrastructure offered by XML directly. For instance, one can use XPath to query a FoLiA document and XSL to transform it. To do so effectively, however, the user/developer needs to be more familiar with the intricacies of FoLiA, than when using a tool from the FoLiA infrastructure that abstracts over this for the benefit of the user/developer.

Many of the tools of the core layer are available as command-line tools and are bundled in two software packages: there is a Python-based **FoLiA Tools** package[1] and a **FoLiA Utilities** package[2] consisting of tools written in C++. Both are built on the respective libraries. There is some overlap in tools, but each also offers distinct tools the other does not. It is therefore recommended to install both. These packages, and all others tools pertaining to the FoLiA infrastructure developed at Radboud University, are also bundled in our **LaMachine** distribution.[3]

We subscribe strongly to the CLARIN principle that standards should be open and place a similar requirement on the infrastructure components we build.

[1] https://pypi.python.org/pypi/FoLiA-tools

[2] https://github.com/LanguageMachines/foliautils

[3] http://proycon.github.io/LaMachine/, available as a Virtual Machine, Docker package or local installation script

4.1 Programming Libraries

At the heart of the FoLiA infrastructure are the *programming libraries* that enable developers to work with documents in the format in their software. We ourselves offer libraries for both Python and for C++.

Python is a widely popular high-level programming language in the academic world, and the NLP world in particular. The Python library for FoLiA enables develops to quickly integrate support for FoLiA in their scripts. The library is part of the larger **PyNLPl** library[4] and is also available from the Python Package Index. [5] It is extensively documented and comes with tutorials for users.

The Python library suffers from the performance drawback that any high-level interpreted language has. Whenever faster processing is required, or integration in high-performance tools is desired, **libfolia**[6], the FoliA library for C++, offers a better solution. The library is modelled after the Python library, so both are similarly structured, employ a similar syntax and the respective authors try to do their best to keep the libraries in sync.

A third popular language in the field is Java, but there is no Java-based FoLiA library is available yet to our knowledge. Nevertheless, there are a number of java-based tools in the FoLiA infrastructure that have been developed without a common underlying FoLiA library.

4.2 Validation

We already touched upon the notion of shallow and deep validation. FoLiA's syntax is formalized in a RelaxNG schema, and shallow validation can therefore be done using any XML validator with support for RelaxNG.

The tools **foliavalidator** and **folialint**[7] also perform shallow validation, and their usage is strongly recommended, or should even be considered mandatory, for anybody producing FoLiA documents. Moreover, the former tool can optionally perform deep validation as well, i.e. it can validate the used classes against the set definitions.

4.3 Conversion

The FoLiA tools & utilities collections contain tools for the conversion from and to various different other formats:

- Conversion to plaintext
- Conversion to HTML
- Conversion to simple columned data or CSV
- Conversion from/to ReStructuredText[8]
- Conversion from/to DCOI XML format (Apperloo, 2006)
- Conversion from the Alpino XML format (Bouma et al., 2000)
- Conversion from ALTO DIDL format[9]
- Conversion from hOCR HTML format (Breuel, 2007)
- Conversion from Page XML format[10]

[4] https://github.com/proycon/pynlpl
[5] https://pypi.python.org/pypi/PyNLPl
[6] https://github.com/languagemachines/libfolia
[7] Part of respectively **foliatools** and **foliautils**
[8] `http://docutils.sourceforge.net/rst.html`
[9] `http://www.loc.gov/standards/alto/`
[10] `http://www.primaresearch.org/tools`

Conversions may be limited by the source or target format. Conversion to FoLiA's predecessor DCOI XML, for instance, is only possible for the subset of elements that DCOI supports. Similarly, conversion to ReStructuredText is limited to text, its structure and markup, and does not include linguistic annotations.

Besides the in-house developed FoLiA tools, third parties also make available converters from or to FoLiA. A notable case is **OpenConvert**[11], developed by the Institute for Dutch Lexicology (INL), which can convert from TEI, plaintext, Alto, Microsoft Word, and HTML to FoLiA.

4.4 Visualisation

An XSL stylesheet is available to visualise FoLiA documents. It renders documents and unobtrusively pops up with annotation information when hovering over structural items such as words. A major advantage is that this form of visualisation can be conducted entirely client-side in nearly every web browser. The **folia2html** conversion tool also employs the same stylesheet.

4.5 Searching

Tools for searching and querying FoLiA documents can be divided into two categories:

1. In-document search;
2. Document retrieval systems / Corpus Search tools.

At a low level, in-document search can be conducted with the command-line tool **foliaquery**, part of the FoLiA tools. This tool reads one or more FoLiA documents in memory (sequentially), executes a search query, and presents the matching results. This, however, is not a solution that scales to large numbers of documents as it takes a fair amount of time and memory to process a document.

Full document retrieval systems do not rely on such costly real-time processing of the FoLiA documents, but construct smart indices from the original documents and operate on these indices. The software **Blacklab** (back-end) and (front-end) **Whitelab** (Reynaert et al., 2014) are examples of this. They were constructed in the CLARIN-NL project OpenSoNaR, and can operate on FoLiA documents. It has to be added though, that unlike the low-level in-document search, such engines typically only support a simpler subset of the annotation types supported by FoLiA, such as Part-of-Speech tags and lemmas. At this moment, span annotation types such as dependency relations, syntax and named entities, are not supported yet.

As FoLiA is a highly expressive format, the need arose for a query language tuned specifically to the idiosyncrasies of FoLiA. Although FoLiA can be perfectly searched with XPath, formulating a robust query is not always trivial and may require more in-depth knowledge of FoLiA. The *FoLiA Query Language* (FQL) was designed as a higher-level query language, covering all of FoLiA, to make querying FoLiA documents easier. FQL is implemented alongside the FoLiA Python library in **PyNLPl**[12]. It is documented as part of the FoLiA documentation (van Gompel, 2014).

FQL is a new and expressive query language. People in the field may be more accustomed to simpler and established query languages such as the *Corpus Query Language (CQL)* (Chris and Schulze, 1996), developed at the Corpora and Lexicons group, IMS, at the University of Stuttgart in the early 1990s. For this reason, **PyNLPl** includes a library that converts CQL to the more expressive but verbose FQL has been implemented as well. The low-level query tool makes use of both these libraries. In the next section we will discuss FQL further and introduce higher-level tools in the FoLiA infrastructure that make use of it.

4.6 Editing

FQL has been designed in such a way that it is not just a language for passive querying, but a language that allows active manipulation of FoLiA documents. In other words, FQL is to FoLiA as SQL is to

[11] https://github.com/INL/OpenConvert
[12] https://pypi.python.org/pypi/PyNLPl/

relational database tables. Therefore, the **foliaquery** command-line tool and the FQL library it relies on can not just be used to passively retrieve information, but also to actively edit documents.

A FoLiA document server [13] has been constructed as a back-end for the editing of FoLiA documents. It is implemented as a RESTful webservice, with a simple human-interface to manually enter queries, and takes care of on demand loading and unloading documents in memory and serialising them to disk. It maintains a browsable document repository, which features **git** version control support.

Neither the command-line tool nor the document server offer an interface adequate for human end-users to easily work with. To provide such an environment, we have been developing the **FoLiA Linguistic Annotation Tool** (**FLAT**)[14]. It is a modern web-application that offers an interface for the visualisation and editing of FoLiA documents. Under the hood, user-interface interactions are translated to FQL queries and communicated to the aforementioned FoLiA document server.

Although not yet supporting all of FoLiA at the current stage, **FLAT** has already been used succesfully in several annotation projects featuring student assistants at Radboud University. Further development of FLAT is planned for the CLARIN-NL successor project CLARIAH, with the aim of providing a mature editing environment covering all of FoLiA.

4.7 Special-purpose tools

The previous sections discussed tools that can be considered part of the core layer. In this section we will discuss the outer layer of tools, these are tools that either take FoLiA as their input or deliver it as their output to perform a specific and specialised task, usually an NLP task given our context. It is a most essential layer to the infrastructure.

- **Ucto**[15] – An advanced rule-based tokeniser for a variety of languages. Supports FoLiA input and output. (van Gompel et al., 2012)
- **Frog**[16] – An NLP suite for Dutch, implementing tokenisation (through ucto), Part-of-Speech tagging, Lemmatisation, Dependency Parsing, Named Entity Recognition, Shallow parsing and Morphological Analysis. Supports FoLiA input and output.
- **CLAM**[17] – Turns command-line NLP tools into RESTful webservice with an interface for human end-users. It integrates the FoLiA viewer to visualise FoLiA documents. (van Gompel, 2012)
- **Ticcl** – Text-Induced Corpus Clean-up. Supports FoLiA input and output. Used in the CLARIN-NL projects TICCLops and @PhilosTEI. (Reynaert, 2010)
- **Gecco**[18] – Generic Environment for Context-Aware Correction of Orthography: A spelling correction engine fully based on FoLiA. Powers *Valkuil.net* and soon also *Fowlt.net*.
- **T-Scan**[19] – A Dutch text analytics tool for readability prediction. (Maat et al., 2014)
- **Cesax**[20] – A co-reference editor for syntactically annotated XML corpora. Supports FoLiA import and output through conversion.
- **Colibri Core**[21]– A tool for the computation of corpus statistics on ngrams and skipgrams in a quick and memory-efficient way. It can import FoLiA documents, which it subsequently compresses to an internal optimised binary format.

[13] https://github.com/proycon/foliadocserve
[14] https://github.com/proycon/flat
[15] https://languagemachines.github.io/ucto
[16] https://languagemachines.github.io/frog
[17] https://proycon.github.io/clam
[18] https://github.com/proycon/gecco
[19] https://github.com/proycon/tscan
[20] http://erwinkomen.ruhosting.nl/software/Cesax/
[21] https://proycon.github.io/colibri-core

5 Data Infrastructure

A format's usefulness is not just determined by the tools available, but also by the data sets delivered in the format. The following corpora are currently delivered in FoLiA:

- **Basilex** – The Basilex corpus collects Dutch written language by children, and contains about 11.5 million words. It includes lexical semantic sense annotation.(Tellings et al., 2014)
- **DutchSemCor** – The DutchSemCor project delivered a Dutch corpus annotated with lexical semantic senses. Part of the annotation was manual, and a part was tagged automatically with Word Sense Disambiguation system trained on the manual part. The corpus is based on SoNaR, as well as extra sources. (Görog and Vossen, 2010)
- **SoNaR-500** – The STEVIN project SoNaR delivered a 540 million word corpus of written Dutch (including Flemish) from numerous sources. The corpus is annotated with Part-of-Speech tags, lemmas, and named entities. (Oostdijk et al., 2013)
- **Nederlab** – The Nederlab project attempts to collect all digitized texts relevant to the history of Dutch language, culture and heritage (circa 800 – present) in one user friendly and tool enriched open access web interface. (Brugman et al., 2016).
- **VU-DNC** – A diachronic Dutch newspaper corpus (2 million tokens) with annotations of subjectivity and providing a gold standard for OCRed newspapers published in 1950. (Vis et al., 2012)

In addition to corpora, the data part of the infrastructure also consists of a growing number of Set Definitions.[22]

6 Conclusion

In this paper we have described the rich infrastructure that has been developed around the Format for Linguistic Annotation (FoLiA). We emphasised the need for a practical and proven format, in line with CLARIN's standardisation principle, and hence placed the focus for this paper on the software and data infrastructure. A more extensive overview of FoLiA itself and motivation for its inception was presented in earlier work (van Gompel and Reynaert, 2013).

Continued efforts in the CLARIN-NL successor CLARIAH ensure that the developments on the infrastructure surrounding FoLiA will continue in the foreseeable future.

References

Wilko Apperloo. 2006. XML basisformaat D-Coi: Voorstel XML formaat presentational markup. Technical report, Polderland Language and Speech Technology.

Gosse Bouma, Gertjan van Noord, and Rob Malouf. 2000. Alpino: Wide-coverage computational analysis of Dutch. In Walter Daelemans, Khalil Sima'an, Jorn Veenstra, and Jakub Zavrel, editors, *CLIN*, volume 37 of *Language and Computers - Studies in Practical Linguistics*, pages 45–59. Rodopi.

Thomas Breuel. 2007. The hOCR microformat for OCR workflow and results. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition*, volume 2, pages 1063–1067. IEEE Computer Society.

Daan Broeder, Oliver Schonefeld, Thorsten Trippel, Dieter Van Uytvanck, and Andreas Witt. 2011. A pragmatic approach to XML interoperability the Component Metadata Infrastructure (CMDI). In *Balisage: The Markup Conference 2011*, volume 7.

Hennie Brugman, Martin Reynaert, Nicoline van der Sijs, René van Stipriaan, Erik Tjong Kim Sang, Antal van den Bosch, Jan Pieter Kunst, Rob Zeeman, Dieuwertje Kooij, Ineke Brussee, Matthijs Brouwer, Marc Kemps-Snijders, and Hans Bennis. 2016. Nederlab: Towards a single portal and research environment for diachronic Dutch text corpora. In Nicoletta Calzolari et al., editor, *Proceedings of the Tenth International Language Resources and Evaluation Conference (LREC-2016)*, Portoroz, Slovenia. ELRA.

[22] https://github.com/proycon/folia/tree/master/setdefinitions

Oliver Chris and Bruno M. Schulze. 1996. Ein flexibles und modulares anfragesystem für textcorpora. In *Lexikon und Text*, pages 121–133. Max Niemeyer Verlag.

Attila Görog and Piek Vossen. 2010. Computer assisted semantic annotation in the DutchSemCor project. In *Proceedings of the Seventh International Conference on Language Resoruces and Evaluation, LREC-2010*, pages 1220–1226, Valletta, Malta.

Henk Pander Maat, Rogier Kraf, Antal van den Bosch, Nick Dekker, Maarten van Gompel, Suzanne Kleijn, Ted Sanders, and Ko van der Sloot. 2014. T-scan: a new tool for analyzing dutch text. *Computational Linguistics in the Netherlands Journal*, 4.

Nelleke Oostdijk, Martin Reynaert, Véronique Hoste, and Ineke Schuurman. 2013. The construction of a 500-million-word reference corpus of contemporary written Dutch. In *Essential Speech and Language Technology for Dutch: Results by the STEVIN-programme*, chapter 13. Springer Verlag.

Martin Reynaert, Matje van de Camp, and Menno van Zaanen. 2014. OpenSoNaR: user-driven development of the SoNaR corpus interfaces. In *Proceedings of COLING 2014: System Demonstrations*, pages 124–128, Dublin, Ireland. Dublin City University and Association for Computational Linguistics.

Martin Reynaert. 2010. Character confusion versus focus word-based correction of spelling and OCR variants in corpora. *International Journal on Document Analysis and Recognition*, 14:173–187. 10.1007/s10032-010-0133-5.

Agnes Tellings, Micha Hulsbosch, Anne Vermeer, and Antal van den Bosch. 2014. Basilex: an 11.5 million words corpus of dutch texts written for children. *Computational Linguistics in the Netherlands Journal*, 4:191–208, 12/2014.

Maarten van Gompel and Martin Reynaert. 2013. Folia: A practical xml format for linguistic annotation - a descriptive and comparative study. *Computational Linguistics in the Netherlands Journal*, 3.

Maarten van Gompel, Ko van der Sloot, and Antal van den Bosch. 2012. Ucto: Unicode Tokeniser. Reference Guide. Technical report, Tilburg Centre for Cognition and Communication, Tilburg University and Radboud Centre for Language Studies, Radboud University Nijmegen.

Maarten van Gompel. 2012. CLAM: Computational Linguistics Application Mediator. documentation. Technical report, Tilburg Centre for Cognition and Communication, Tilburg University and Radboud Centre for Language Studies, Radboud University Nijmegen.

Maarten van Gompel. 2014. Folia: Format for linguistic annotation. documentation. Technical report, Radboud University Nijmegen.

Kirsten Vis, Jos Sanders, and Wilbert Spooren. 2012. Diachronic changes in subjectivity and stance – a corpus linguistic study of dutch news texts. *Discourse, Context & Media*, 1(23):95 – 102. The view from here, there and nowhere: discursive approaches to journalistic stance.