Adapting a Web Application for Natural Language Processing to Odd Text Representation Formats

Bart Jongejan

Department of Nordic Studies and Linguistics
University of Copenhagen, Denmark
Email: bartj@hum.ku.dk

***Abstract*—Users of Natural Language Processing (NLP) are best helped if that technology has a low threshold. Therefore, there is a niche for NLP infrastructures that can adapt to the notations used in scholarly projects, instead of requiring that projects adapt to the notation prescribed by a particular NLP infrastructure. The Text Tonsorium is a web application that fits in that niche, because it is not married to any notation and therefore can integrate tools that are tailored to the needs and notations of projects. There are no costs involved related to manually modelling project specific tools into workflow templates, since the Text Tonsorium automatically computes those templates.**

Keywords–Natural Language Processing; NLP workflows; digital edition; medieval diplomas.

I. INTRODUCTION

A. Notations, encodings, file formats

Researchers use notations to express their thoughts and findings in ways that can be understood by their peers. Examples are Venn diagrams, staff notation (for music), Arabic numerals, and notations for regular expressions. Incompatible notations, such as Arabic and Roman numerals, can and do live alongside each other. That is neither good nor bad, but just a reality. The use of different notations for the same thing is sometimes a precondition for progress.

In the computer age, the related concepts of encoding and file format have also become prominent. For software to be able to automatically add annotations to a scholarly document, say, the software has to "understand" the encoding, the file fomat, as well as the notation of the input. In this paper, the distinctions between these three concepts are not important. "Notation" will be used as the generic term for all the conventions that have to be adhered to in order to make successful use of software.

B. NLP infrastructures prescribe notation

There is a tendency in Natural Language Processing (NLP) infrastructure projects to strive for notations that are adhered to by all who want to use the services of those infrastructures. The adoption of widely used notations is probably good for a large number of projects. However, there are also scholarly projects that decide to use notations that primarily achieve other goals than the option to use NLP, for example, that it must be possible to make manual annotations in a visually attractive way, or that project participants do not have to be retrained. In addition, adopting the notation prescribed by an NLP infrastructure has a risk. The project may bet on the wrong notation by choosing a specific NLP infrastructure: notations promoted by infrastructures proliferate at the same rate as projects implementing those infrastructures and can become obsolete after a short time. Universal notations that can replace all other notations and that are not only safe to use now, but also in the foreseeable future, do not exist. Conversely, not adhering to the notation required by an NLP infrastructure excludes scholarly projects from the use of that infrastructure.

C. Mapping between notations

When making NLP tools available to a scholarly project that uses its own notation, there is a need for a technical mediation between the notation employed in that project and the notations required by the NLP tools that currently are in the toolbox. Ideally, the mapping between the notation employed by the project and the notation used by existing NLP tools goes both ways, so that results from the NLP infrastructure appear in the notation employed by the project.

One way to realize a mapping is to let the scholarly project be responsible for the necessary conversions, so that no adaptation of the NLP infrastructure is necessary. This approach was, for example, adopted in the DK-Clarin project [1]. The goals of this project were twofold: a repository with many Danish linguistic resources, and an on-line NLP service for the Danish language. In order to optimize the usefulness of shared resources, users of the DK-Clarin repository were requested to only contribute text resources that complied with a particular schema, called "TEIP5 DK-CLARIN", that followed the Text Encoding Initiative guidelines, version P5 (TEI P5). This notation was agreed on by the users who participated in the DK-Clarin project. The expectation was that other users would also adopt this notation. To nudge users in the right direction, it was decided that the NLP tools could only be applied to resources that had been deposited in the repository. The idea was that the cost of transition from non-conforming notations to the "TEIP5 DK-CLARIN" notation would be outweighed by the advantage of being able to use the NLP tools. In that way, the infrastructure would not have to carry the cost of conversion of notation, but could turn that over to the users.

D. Structure of the paper

The structure of the remaining part of this paper is as follows. Section II presents a workflow management system, the Text Tonsorium (TT), that does not prescribe the use of any particular notation and therefore can be used in projects that use their own notation. Section III presents related work. Section IV presents a use case that illustrates how the TT can adapt to a project that uses its own notation. This is done step

 ISBN: 978-1-61208-706-1

by step in four subsections: upload of data, specification of the desired output, computation and selection of workflows, and tracking the execution of the selected workflow as it progresses through its job steps. Section V describes how the Directional Acyclic Graph (DAG) structure of workflows makes it possible to handle both common and project specific notations. That section also describes in a few words how the TT computes workflows. Section VI gives a short outline of the human efforts that are needed to integrate a tool. The concluding remarks and future plans are drawn in Section VII.

II. The Text Tonsorium

This paper exposes the benefits of an approach that is almost opposite to the approach that gives users of an NLP infrastructure the sole responsibility for necessary notation conversions. Both projects and the NLP infrastructure gain something by supplementing an existing and evolving NLP infrastructure with project specific tools for handling project specific notations. Projects thus have the advantage that the notation mapping problem is solved by the NLP infrastructure, while, in the wake of the adaptations for project specific purposes, the infrastructure may very well be enriched with tools that are also useful for a general public. Also the quality of the output from the NLP tools may improve, since the cooperation between a project and the NLP infrastructure may produce new or improved linguistic resources with which NLP tools like Part of Speech taggers and lemmatizers can be (re-)trained.

A precondition for the viability of this approach is that the cost of extending the infrastructure with project specific tools is manageable. Most importantly, the cost per extension should stay more or less constant. The TT is a workflow manager for NLP that supports this approach and does so at a cost that is manageable, because the cost of integration of a new tool does not depend on the number of already integrated tools.

The cost of integration of a new tool would hardly remain the same as the number of already integrated tools grows if existing, preconfigured workflow templates would have to be copied and manually adapted to new notations, since the number of such workflow templates very likely also would grow and the average workflow template would become more complex. The TT eliminates the manual construction of workflow templates. Instead, it creates workflow templates automatically, basing its computations on the features of the actual input, the user's requirements with respect to the output, and the metadata of the tools that are registered in the infrastructure.

The TT was built during the DK-Clarin project as the NLP component of the Clarin.dk [2] infrastructure. Two requirements determined the architecture of this component. Both requirements had the purpose of minimizing the maintenance effort needed to run the infrastructure, so that it could continue to be available and growing in times of low funding.

The first requirement was that if a user of the Clarin.dk infrastructure would like to share a tool with other users, the registration and integration of that tool had to be done by the user, and not by the maintainers of the TT. The second requirement was that the maintainers of the TT should not be involved in the construction of workflow templates.

The second requirement could have been fulfilled by just not offering facilities for workflows at all or by offering a facility that would enable users to construct workflows by hand. The choice fell on a solution that required a user interface with only few fields and controls, and a back-end that computed viable workflow templates automatically, using the characteristics of the input and the desired output as the boundary conditions for the computation of workflow templates.

The possibility to handle other notations than the "TEIP5 DK-CLARIN" notation was a fortuitous side effect of this architecture, but it was not a publicly accessible feature until, in 2017, the NLP component became a web application [3] independent of the Clarin.dk repository, under the new name "Text Tonsorium".

A detailed technical description that explains how the TT computes workflow templates and why most of the implementation was done in a domain specific programming language for Symbolic Mathematics, Bracmat, is in [4]. More about the user perspective of the TT is in [5]. The TT is open source [6].

III. Related Work

Most NLP workflow management systems require (expert) users for the construction of workflow templates and either have elaborate graphical interfaces and tool-profile matching algorithms to assist the user, or require that the user does some scripting.

Weblicht [7] offers NLP workflows that can take a range of file formats as input but no resources that already have some structure, such as metadata and manually created annotations in the text that should not get lost in the NLP workflow. The exception are resources expressed in the Text Corpus Format (TCF) [8], which combines several stand-off annotation layers in a single file. The TCF is the native file format in Weblicht and is used for all data interchange between the tools.

The Nextflow system [9][10] powers the Dutch Philosophical Integrator of Computational and Corpus Libraries (PICCL) [11] portal. The Format for Linguistic Annotation (FoLiA) [12] is the standard notation in PICCL.

Other systems that require the manual construction of workflow templates are, for example (in alphabetical order): Galaxy [13], Gate [14], Kathaa [15], Kepler [16], Taverna [17], TextGrid [18], UIMA [19], and zymake [20].

Curator [21] is a workflow management system with a limited set of NLP tools. Some of these tools depend on outputs from other tools, yet manual construction of workflow templates is not necessary. Workflows are implicitly and uniquely defined in Curator because there is exactly one tool for each type of annotation layer.

Universal Dependencies (UD) is an international effort to develop parsers for many languages. Software contributions must conform to the notation defined for the Conference on Computational Natural Language Learning (CoNLL). [22]

IV. Use case: add PoS tags and lemmas to transcriptions of medieval diplomas

This section shows how the TT adds Part of Speech tags and lemmas to documents that use a project specific notation. The project, *Script and Text in Space and Time* (STST) [23], has the goal, among other things, to provide dynamic and interactive digital editions of medieval diplomas.

The TT is in a way similar to software that computes routes between two addresses on a map. From the user's perspective, there are four steps. On the front page, the user is invited to

 ISBN: 978-1-61208-706-1

upload input for NLP processing. On the second page, the user specifies the output. On the third page, the user selects a workflow from a list of possible workflows. On the fourth and final page, the user can follow the execution of the workflow as it progresses, and see the outputs.

The enhancements to the TT that were made for the sake of the STST project will be pointed out as we discuss each of the four steps.

A. *First step: upload data*

The front page of the TT is shown in Figure 1. The user has three options to send input to the TT: by upload of files, by listing Universal Resource Locators (URLs), and by direct text entry. These methods can in principle be combined, but the TT currently assumes that all uploaded sources have the same language, file type, type of content, etc. The upload starts when the user presses the *Specify the required result* button.

Since, in principle, data that is uploaded to the TT could pass through NLP tools that are monitored by third parties, the user is asked not to upload sensitive data.

Figure 1. Front page of Text Tonsorium with three input modes.

In this example, the user uploads a single file, “24.org” [24]. This file contains a header and a table, and utilizes Org-mode [25], a notation native to the editor of choice in the STST project, Emacs. The project uses a GitHub repository as shared work space for this and hundreds of other transcriptions of diplomas. For the STST project, an extra benefit of using GitHub is that it has provisions for visualizing Org-mode files in an attractive way.

B. *Second step: specify the desired output*

When the input is uploaded, the TT’s first action is to find out what it is. In this example, the input is uploaded with the media type “application/x-download”, “application/octet-stream”, or “text/plain”, depending on the user’s browser. Since these media types are very general, the file is opened by the TT and its content analyzed. The TT ascertains, for example, whether a text file conforms to the aforementioned project’s notation. About 360 characters of code are dedicated to this specific analysis. The result of the analysis is shown in the upper part of the second window, see Figure 2. In the shown example, the TT was able to fill out all fields. In general, the TT does not know the language of the input and leaves that field empty, but in this case the language, Latin, is revealed in the header section of the uploaded file.

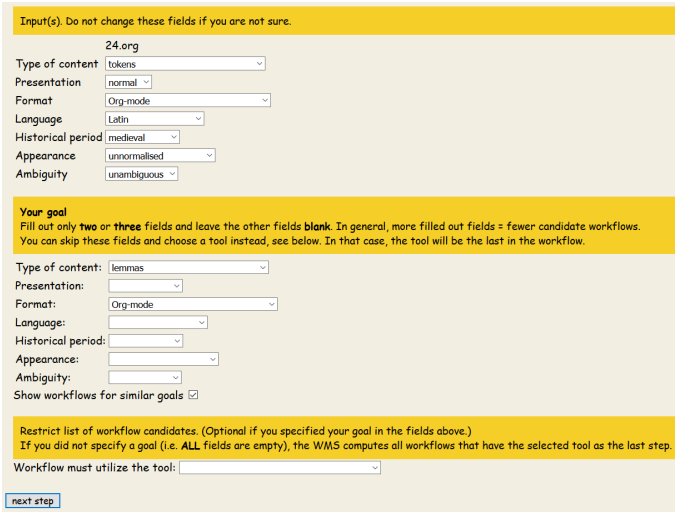

Figure 2. Window where the user specifies the output: lemmas, Org mode. The input features are set by the Text Tonsorium.

The lower part of the window in Figure 2 is where the user specifies the goal. The goal, like the input, is specified in terms of one or more features.

Currently seven features can be specified, and that number can change in the future. Originally, there were only three features, namely those for *language*, *file format* and *type of content*. Later came *presentation* (indicating whether or not a result was sorted, and if yes, alphabetically or according to frequency), *appearance* (whether a result was “noisy”, “human readable” or just “clean and concise”), *historical period* (“classical”, “medieval”, “early modern”, “late modern” and “contemporary”), and *ambiguity* (“unambiguous”, “ambiguous”, or “pruned, but not necessarily unambiguous”).

Some feature values can subsume other values as well. For example, the *type of content* called “lemmas” also includes the combination “segments, lemmas”, which means that sentence structure of the input is still intact in the output.

The user is advised to leave some fields empty in the goal specification. A very detailed specification decreases the chances that any workflow can fulfil the goal. A good strategy is always to specify the language (this can also be done in the input) and the type of content, and perhaps also the format.

More expert users of the TT can optionally specify a tool that the workflow has to contain. If the output fields are empty, then the output specifications of the selected tool are taken as the goal, and the selected tool will be the last in the workflow. If some of the output fields are also filled out, then the selected tool can be anywhere in the workflow.

In this example the user specifies that the output must have lemmas and that it must have the Org-mode file format. Because the *Show workflows for similar goals* field is checked, the TT will also try to fulfil goals that offer Part of Speech tags besides lemmas, and a few other goals.

C. *Third step: workflow templates are computed and user selects one*

When the user presses the *next step* button on the output specification page, the TT starts to compute workflow templates that, given the current input, lead to the goal specified by the user. If no workflow template exists that can fulfil the goal with the tools that are currently integrated, then the TT

 ISBN: 978-1-61208-706-1

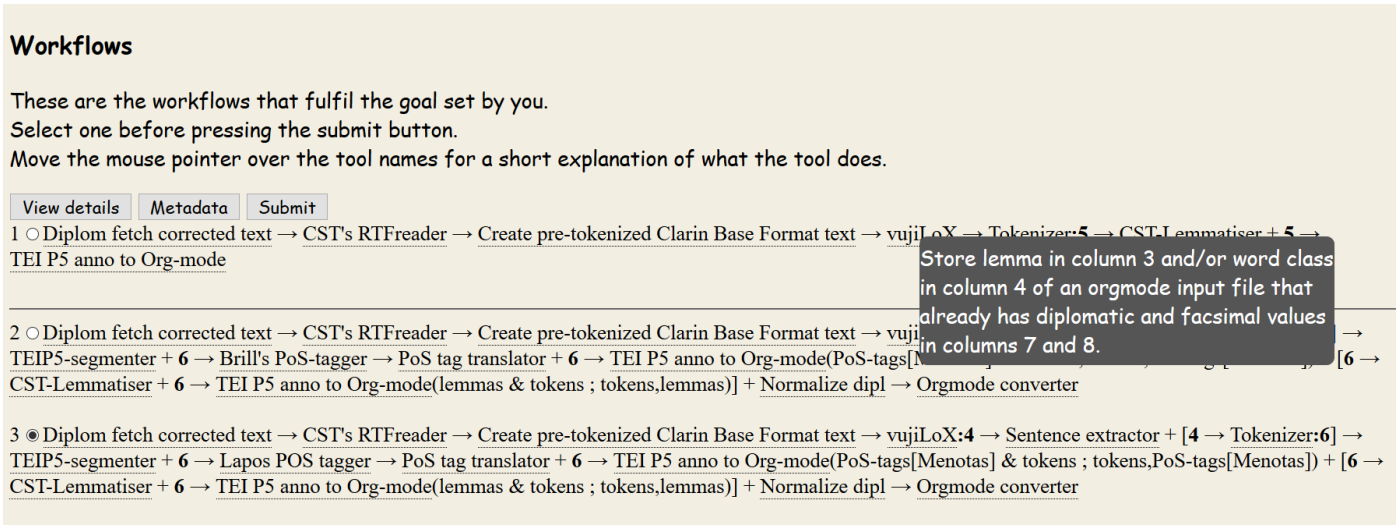

Figure 3. The three workflows that lead to the user's goal. The user has already chosen the third workflow. The "Orgmode converter" tool is explained.

will quickly tell the user that. On the other hand, if there are workflows, then the best ones will be listed.

The list can be quite long. If that is the case, the user is perhaps able to see that the list in part is populated by workflows that are meant for the wrong historical period, or that deliver ambiguous output, or that have other undesirable common features. She can then go back to the output specification window and specify the output in more detail by leaving fewer fields empty. In that way, it is often possible to reduce the presented list to such a degree that the user gets a well organized overview over the possibilities.

Quite often the user will see workflows that are the same apart from different styles of some feature values. For example, the *content type* called "tags" has several tag styles, such as "Universal" and "Penn Treebank" (provided that the language is English). Another example are the two different styles of the "html" value of the *format* feature, one style saying that the body uses the traditional `h`, `p`, `table`, `br`, etc. elements, and another style that does not involve these elements. The latter style is hard to process in an NLP workflow, but defines the text layout to an extremely high degree, as in a PDF document. Normally style values are kept out of sight of the user. Styles are hard to specify for non-specialist users and would add a lot of unintelligable clutter to the user interface.

In this example, the TT lists three workflows, see Figure 3. The first workflow does not involve a Part of Speech tagger, so the user can discard it. The second and third workflows are very similar. The only difference is the Part of Speech tagger. The second workflow involves a Brill POS-tagger, while the third workflow involves the Lapos POS-tagger.

Common to all three workflows and specifically implemented to meet the special needs in the STST project are the tools "Diplom fetch corrected text", "Normalize dipl" and "Orgmode converter". These three tools handle Org mode files that have the internal organization used in the project. The tools "vujiLoX" and "TEI P5 anno to Org-mode" were also created for the sake of this project but are of a general nature. The "vujiLoX" tool lowercases all characters in the input and also converts all *v* to *u* and all *j* to *i*. This tool prepares Latin texts for the lemmatiser. The "TEI P5 anno to Org-mode" tool combines two stand-off annotations into a single two-column table in Org-mode notation. These two tools show that the TT's involvement in a project not only helps the project but can also be useful for users outside that project.

D. Fourth step: execution of the selected workflow and viewing the output

Once the user has selected a workflow and pressed the *next step* button, the TT will execute the selected workflow for all the inputs that the user has uploaded.

Figure 4. The third workflow halfway being executed.

The output of a tool can be viewed as soon as the tool finishes. The user can reload the page where the workflow unfolds or wait for the automatic page refresh that comes every 10 seconds, see Figure 4.

When all processes in a workflow have been executed, all results can be downloaded in a single zip file. The results can be downloaded again, but after a few days, they are deleted from the server. The output of the workflow is, in this example, the input file enriched with values in columns that were originally empty, see Figure 5.

V. The Structure and Computation of Workflows in the Text Tonsorium

The TT follows the dataflow programming paradigm, which means that a workflow template computed by the TT has the layout of a DAG (See Figure 6) and that NLP tools in mutually independent paths of the graph can be executed at the same time. For example, in Figure 4, the "CST-lemmatiser"

:m:		
Text number	24	
Shelfmark	AM dipl dan fasc LI 11	
Year	1257	
Date	Jan 13	
Place	Lateranet	
Language	lat	
Scribe		
Material	perg	
Sender	Pave Alexander 4.	
Jexlev	9	
Rep.	247	
Dipl. Dan.	2 rk. I nr. 205	
Reg. Dan.		
Æa.	580	
SDHK		

*** Notes

*** Transcription

:s:						
pm	lr					
lm	24-01					
PE	b					
w		alexander	PROPN	alexander	ALEXANDER	ALEXANDER
PE	e					
w		episcopus	NOUN	episcopus	ep(iscopu)s	ep&pk;s
w		seruus	NOUN	seruus	seruus	feruus
w		seruus	NOUN	seruorum	seruor(um)	feruoꝝ
w		deus	NOUN	dei	dei	ðeı
p		/	X	/	/	/
w		dilectus	VERB	dilectis	Dilectis	Dıleƈtis
w		in	ADP	in	in	ıı̅
w		christus	PROPN	christo	(Christ)o	xp&pk;o
w		filia	NOUN	filiabus	filiab(us)	fılıabꝫ
p		..	PUNCT	..	..	..
w		abbato	VERB	abbatisse	Abbatisse	Abbatıffe
w		et	CCONJ	et	et	et
w		conuentus	VERB	conuentui	Conuentuj	Conuentuı
w		monasterium	NOUN	monasterii	Monasterij	Monaſterij

Figure 5. Part of the output in Org-mode. Columns 3, 4 and 6 contain results from three tools. All other content is copied from the input.

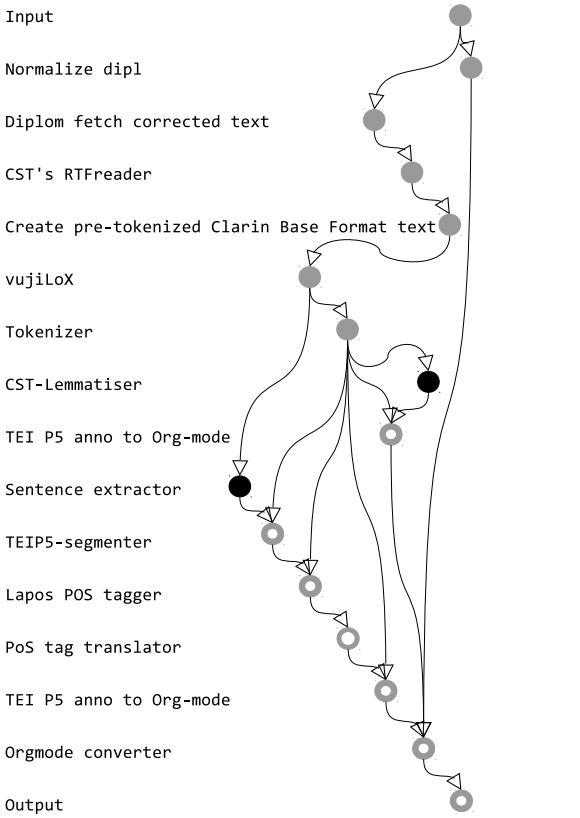

Figure 6. Topological ordering of the third workflow. The black nodes are the currently running steps in Figure 4.

and the "Sentence extractor" are both being executed. These two tools are marked with black filled circles in Figure 6.

The TT computes DAGs by dynamic programming, followed by a pruning step. It starts by fulfilling the user's goal by the output specifications of some tool. It then defines the specifications of that tool's input(s) as the new goal(s). The process is repeated until all goals are satisfied by the specifications of the user's input. Any DAG that leads from the user input to an intermediate goal is memoized, saving both memory and time if the same intermediate goal is needed to satisfy the needs of another tool in the completed workflow template. After a complete DAG is found, the TT backtracks and does an exhaustive search for alternative DAGS, using different tools or the same tools with different settings. The number of found DAGs, which can run in the thousands, is afterwards reduced by excluding those DAGs that have characteristics that users very likely would regard as erroneous, such as multiple occurrences of a particular tool that have different parameter settings for no good reason.

Before presenting the pruned list of workflow template candidates for the user, the TT suppresses details that are not essential for seeing the differences between the candidates. For example, in Figure 3, information about the language, which is Latin in all three workflows, is not shown.

Before a workflow is executed, the corresponding workflow template is instantiated with the actual input provided by the user, and expressed as a list of job steps. Each step comprises the URL of the tool to run, the specification of the necessary input(s) (user provided input or output from other steps), and the actual values of the input and output parameters. Each time new output becomes available, the TT activates job steps for which all required inputs are available.

A huge advantage of DAG-structured workflows is that they can involve both tools that are aware of special notations and tools that are not. Though not impossible, this is hard to achieve with workflows that have a strictly linear structure, as is the case with workflows that use TCF, CoNLL, FoLiA, or other notations that accumulate annotations while traversing a sequence of tools.

The DAG structure has the additional quality that processes in different branches can be executed synchronously.

VI. Integration of tools

There are some limitations as to which types of tool can be integrated in the TT. The tool must run from the command line and may not require any user interaction while running. Also, all parameters to the tool must either be fixed or take values from a nominal scale [26]. This makes it hard or impossible to integrate, for example, Deep Learning scripts that require that the user experiments with several settings of real-valued parameters.

If a candidate tool fulfils these requirements, the TT offers an easy way of embedding it in an ecosystem of already existing tools.

Once a tool is integrated, users of the TT can see that the new tool is taken into consideration when the TT computes workflow templates.

Integration of a new tool starts in the administrative page of the TT. The registration form is in two parts. One part stores name, description, creator, etc., and, most importantly, the URL where the webservice that wraps around the tool can be found. The second part specifies the input and output profile(s) of the tool in terms of features and feature value subspecifications.

After the registration of a new tool, the TT can create a PHP script for the new web service that is already tailored to the new command line tool to be integrated. This PHP script parses the HTTP parameters that the TT will set when the tool is called and fetches all the input files from the TT server that the tool needs. To wrap the script around the tool, the

 ISBN: 978-1-61208-706-1

programmer must look for the pieces of PHP code marked "TODO" and follow the instructions. Dummy code is present that makes it possible to test that the script is callable over HTTP.

VII. Conclusion and Future Work

Three features of an NLP infrastructure are key to being adaptable to scholarly projects that have chosen a notation that is unknown to the NLP infrastructure. The first feature is that it must be cheap to create workflow templates, so that many different projects, each with their own traditions and requirements, can be served at affordable cost. The second feature is that the NLP infrastructure must not impose a notation on projects that want to use the NLP tools, but rather should open up for "odd" notations. The third feature is that the workflow templates should be composable of tools that require varying notations, so that tools that are tailored to specific projects can cooperate with tools that use different notations.

The TT fulfils these three requirements. (1) The cost of integration of new tools is not influenced by the tools that are already integrated, since workflow templates containing many steps are automatically created at no cost. (2) The TT can handle a wide variety of notations in input and in output. (3) The workflow templates that the TT produces are directed acyclic graphs. That makes it straightforward to pass notation around tools that cannot handle it. In the example given in this paper, all the layout and content in the input is reproduced in the output, which is hard to achieve in linear pipelines of NLP tools.

In the future, we want to speed up processing of large amounts of documents by exporting TT's automatically computed workflow templates to faster workflow execution platforms. We also want to improve the user interface by providing much more context sensitive guidance.

References

[1] L. Offersgaard, B. Jongejan, and B. Maegaard, "How Danish users tried to answer the unaskable during implementation of clarin.dk," Nov. 2011, [retrieved: April, 2019]. [Online]. Available: https://cst.dk/dighumlab/publications/dkclarin_SDH_nov2011.pdf

[2] "Clarin.dk," https://clarin.dk/, [retrieved: April, 2019].

[3] "Text tonsorium," https://cst.dk/WMS/, 2017, [retrieved: April, 2019].

[4] B. Jongejan, "Implementation of a workflow management system for non-expert users," in Proceedings of the Workshop on Language Technology Resources and Tools for Digital Humanities (LT4DH). Osaka, Japan: The COLING 2016 Organizing Committee, December 2016, pp. 101–108, [retrieved: April, 2019]. [Online]. Available: http://aclweb.org/anthology/W16-4014

[5] ——, "Workflow management in CLARIN-DK," in Proceedings of the workshop on Nordic language research infrastructure at NODALIDA 2013; May 22-24; 2013; Oslo; Norway. NEALT Proceedings Series 20, no. 89. Linköping University Electronic Press; Linköpings universitet, 2013, pp. 11–20.

[6] "Text tonsorium (source code repository)," https://github.com/kuhumcst/DK-ClarinTools, 2017, [retrieved: April, 2019].

[7] E. W. Hinrichs, M. Hinrichs, and T. Zastrow, "Weblicht: Web-based LRT services for German," in ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden, System Demonstrations. The Association for Computer Linguistics, 2010, pp. 25–29, [retrieved: April, 2019]. [Online]. Available: http://www.aclweb.org/anthology/P10-4005

[8] U. Heid, H. Schmid, K. Eckart, and E. Hinrichs, "A corpus representation format for linguistic web services: The D-SPIN text corpus format and its relationship with iso standards," in Proceedings of the 7th International Conference on Language Resources and Evaluation, 2010, pp. 494–499.

[9] "Nextflow," https://www.nextflow.io/, [retrieved: April, 2019].

[10] P. Di Tommaso et al., "Nextflow enables reproducible computational workflows," Nature Biotechnology, vol. 35, Apr 2017, pp. 316–319, [retrieved: April, 2019]. [Online]. Available: https://doi.org/10.1038/nbt.3820

[11] "PICCL: Philosophical integrator of computational and corpus libraries," https://github.com/LanguageMachines/PICCL, [retrieved: April, 2019].

[12] M. van Gompel and M. Reynaert, "FoLiA: A practical XML format for linguistic annotation - a descriptive and comparative study," Computational Linguistics in the Netherlands Journal, vol. 3, 2013, pp. 63–81, [retrieved: April, 2019]. [Online]. Available: https://www.clinjournal.org/clinj/article/view/26/22

[13] B. Giardine et al., "Galaxy: A platform for interactive large-scale genome analysis," Genome Res, vol. 15, 2005, pp. 1451–1455, [retrieved: April, 2019]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1240089/

[14] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan, "GATE: An architecture for development of robust hlt applications," in Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, ser. ACL '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 168–175, [retrieved: April, 2019]. [Online]. Available: https://doi.org/10.3115/1073083.1073112

[15] S. P. Mohanty, N. J. Wani, M. Srivastava, and D. M. Sharma, "Kathaa: A visual programming framework for NLP applications," in Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations. San Diego, California: Association for Computational Linguistics, June 2016, pp. 92–96, [retrieved: April, 2019]. [Online]. Available: http://www.aclweb.org/anthology/N16-3019

[16] A. Goyal et al., "Natural language processing using Kepler workflow system: First steps," Procedia Computer Science, vol. 80, 2016, pp. 712 – 721, international Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA.

[17] K. Wolstencroft et al., "The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud," Nucleic Acids Research, vol. 41, 2013, pp. W557–W561.

[18] H. Neuroth, F. Lohmeier, and K. M. Smith, "TextGrid - virtual research environment for the humanities," IJDC, vol. 6, no. 2, 2011, pp. 222–231, [retrieved: April, 2019]. [Online]. Available: http://dx.doi.org/10.2218/ijdc.v6i2.198

[19] D. Ferrucci and A. Lally, "Building an Example Application with the Unstructured Information Management Architecture," IBM Syst. J., vol. 43, no. 3, Jul. 2004, pp. 455–475, [retrieved: April, 2019]. [Online]. Available: http://dx.doi.org/10.1147/sj.433.0455

[20] E. Breck, "zymake: A computational workflow system for machine learning and natural language processing," in Software Engineering, Testing, and Quality Assurance for Natural Language Processing. Columbus, Ohio: Association for Computational Linguistics, June 2008, pp. 5–13, [retrieved: April, 2019]. [Online]. Available: https://www.aclweb.org/anthology/W08-0503

[21] J. Clarke, V. Srikumar, M. Sammons, and D. Roth, "An NLP curator (or: How I learned to stop worrying and love NLP pipelines)," in Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), pp. 3276–3282, [retrieved: April, 2019]. [Online]. Available: http://lrec-conf.org/proceedings/lrec2012/pdf/664_Paper.pdf

[22] "Universal Dependencies," http://universaldependencies.org/, [retrieved: April, 2019].

[23] "Script and text in space and time," https://humanities.ku.dk/research/digital-humanities/projects/writing-and-texts-in-time-and-space/, [retrieved: April, 2019].

[24] "Diploma 24.org," https://github.com/Clara-Kloster/Guldkorpus/blob/master/transcriptions/org/working/24.org, 2018, [retrieved: April, 2019].

[25] C. Dominik, The Org-Mode 7 Reference Manual: Organize Your Life with GNU Emacs. UK: Network Theory, 2010, with contributions by David O'Toole, Bastien Guerry, Philip Rooke, Dan Davison, Eric Schulte, and Thomas Dye.

[26] S. S. Stevens, "On the Theory of Scales of Measurement," Science, vol. 103, Jun. 1946, pp. 677–680.