Workflow Management in CLARIN-DK

Jongejan, Bart

Published in:
Proceedings of the workshop on Nordic language research infrastructure at NODALIDA 2013

Publication date:
2013

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Jongejan, B. (2013). Workflow Management in CLARIN-DK. In Proceedings of the workshop on Nordic language research infrastructure at NODALIDA 2013 (Vol. 089, pp. 11-20). Linköpings Universitet: Linköping University Electronic Press. NEALT (Northern European Association of Language Technology) Proceedings Series, Vol.. 20

Workflow Management in CLARIN-DK

Bart Jongejan

Copenhagen University

bartj@hum.ku.dk

ABSTRACT

Clarin.dk, the infrastructure maintained by the CLARIN-DK project, is not only a repository of resources, but also a place where users can analyse, annotate, reformat and potentially even translate resources, using tools that are integrated in the infrastructure as web services. In many cases a single tool does not produce the desired output, given the input resource at hand. Still, in such cases it may be possible to reach the set goal by chaining a number of tools. The approach presented here frees the user of having to meddle with tools and the construction of workflows. Instead, the user only needs to supply the workflow manager with the features that describe her goal, because the workflow manager not only executes chains of tools in a workflow, but also takes care of autonomously devising workflows that serve the user's intention, given the tools that currently are integrated in the infrastructure as web services. To do this, the workflow manager needs stringent and complete information about each integrated tool. We discuss how such information is structured in clarin.dk. Provided that many tools are made available to and through the clarin.dk infrastructure, the automatically created workflows, although simple linear programs without branching or looping constructs, can cover a large swath of users' needs. It is rewarding for both users and tool developers that the infrastructure takes advantage of new tools from the moment they are registered, because there is no need to wait for human expert users to construct and save for later use workflows that incorporate new tools.

KEYWORDS: NoDaLiDa 2013, workflow, tools, automation.

Bart Jongejan 2013. Workflow management in CLARIN-DK. *Proceedings of the workshop on Nordic language research infrastructure at NODALIDA 2013.* NEALT Proceedings Series 20 / Linköping Electronic Conference Proceedings 89: 11–20.

1 Introduction

Using computer software to analyse resources in the field of humanities can be a difficult to attain goal, because software packages often require a good deal of technical stamina. Even if all software is available to a researcher, the prospect of having to deal with technical details may deter many, for example because output from one piece of software not necessarily can be used as input for another piece of software due to a technical issue such as a format mismatch. Very powerful and versatile workflow managers exist that are used to alleviate the burden of finding viable combinations of tools, but usually this is done by assisting the user in selecting tools for each step in a possibly long chain of tools. Even though such workflow managers take care of the fitting-together, users still have to learn to use a workflow editor to program a tool chain. Examples of such workflow managers are WebLicht (Hinrichs et al., 2010), and Taverna (Kemps-Snijders et al., 2012), Kepler and Triana (Funk et al., 2010).

Reuse of existing technology for workflow management in clarin.dk[1] (Offersgaard et al., 2011, Offersgaard et al., 2013) was seen as problematic. WebLicht was mostly text only, whereas the clarin.dk repository had to be populated with resources of many types, some of which had little to do with text, such as videos. Also, from the outset it was decided to work with stand-off annotations, each annotation being a resource in its own right. This design did not match very well with WebLichts preferred TCF-format that combines all annotations in one file that is passed on from tool to tool, getting enriched on its itinerary until the end of the tool chain is reached. Other workflow managers would require that at least one user knew the tools that were integrated and became expert in using the workflow editor. Not before this expert user had made a set of workflows, other, less technically minded users, could reap the fruits of the integrated tools. In a relatively small NLP community like the Danish, this was not an attractive prospect. The minimal requirement was that new tools for any kind of resource could be integrated into the infrastructure without the need to make changes to the infrastructure's software and with the possibility for all users to easily apply tools to the resources of their choice.

2 Baseline for handling workflows

From a user's perspective, our baseline is the web site[2] where we have made available a number of NLP tools for on-line use. Here, researchers and students, but also consultants and IT-people all over the world, can get acquainted with some of our software. This web site is a small user-friendly laboratory where the user can pick tools and chain them into a workflow. Where needed, tools are automatically, yet visibly, added to the tool chain to fulfil the prerequisites of the selected tools further down the road. Only valid combinations of tools can be made, because tools that are incompatible with the current choices are 'greyed out' and made ineligible. See Fig. 1.

[1] https://www.clarin.dk/

[2] http://cst.dk/tools/index.php?lang=en

The interface is attractive because it is kept very simple and transparent at a level that is interesting for the user, while boring details are hidden away. In this way, users are gently drawn into the world of NLP and can do simple experiments without all the technical fuzz.

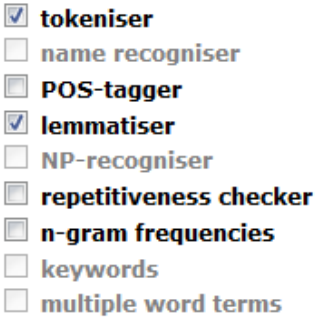

FIGURE 1: The user has chosen the lemmatiser. The system has automatically added the prerequisite tokeniser. The tools in grey text are disabled.

This site has however also its disadvantages. Each new tool has to be integrated by hand, checking out all combinations of tools with the newcomer that are meaningful and above all, disabling all those combinations that aren't. Here, errors are easily made. Another shortcoming is that the web site only takes text, flat or RTF, as input. If we would have to implement support for a wider range of input types, an enormous programming task would lie ahead. And finally, the earlier mentioned focus on tools can also turn into a disadvantage. For many students and researchers the results are what counts and not the tools that are needed to create those results.

3 Tool handling in clarin.dk

Clarin.dk's workflow manager has the same advantages as our base line: tools can be chained into a workflow, tools are automatically added if required by other tools and only valid combinations of tools can be put together in a workflow. There are three big improvements, though. The first is that the system is indifferent to the type of input and output. Not only text, but also pictures, audio and video can be handled. The second improvement is the much easier integration of new tools, which basically takes place by telling the workflow manager how the input must look like, how the output will look like and where the tool, as a web service, is found. This information is stored in a few tables, instead of being expressed in program code as was the case in the baseline. For describing input and output, the tool provider must specify a small number of features by choosing values from predefined lists. If the need arises, an administrator can add values and even features without having to change the workflow algorithms at all. Some feature values can be subspecified, again using any of a number of predefined values. Besides this registration, a tool provider also has to wrap the tool in a web service that is able to communicate with the clarin.dk infrastructure. Because tools are described in a neutral, uniform language, each tool can be seen as isolated from all other tools. Problematic interactions between tools must be solved by correcting the

registered information, or by improving the terminology used for the registration and adapting tools so they better match the expectations as registered in the specifications. For each tool, its service URL and its input/output specification is the only contract with the infrastructure that is vital for employing the tool.

The last and, from a user's perspective, most important improvement is that the user only has to specify a goal by choosing values from a few (four, at the moment) drop-down lists (see Fig. 2.) and can leave it to the workflow manager to find a route to that goal from the given input, in the same way that route planning software finds a route between two locations.

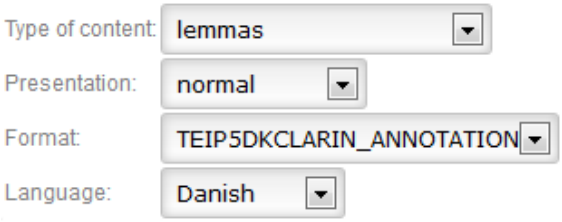

FIGURE 2: Goal specification in clarin.dk

Our approach is reminiscent of the ALPE model (Cristea & Pistol, 2008), which also supports automatic creation of workflows. In contrast to ALPE, however, the I/O specifications in clarin.dk's workflow manager are not hierarchically structured, and the output from a tool is not necessarily an augmented version of the input.

4 Registration of tools

The tool registration facility has two sections, a boilerplate section, and a section dedicated to I/O specification where one or more features are specified. The workflow manager stores all registered tool information in a dedicated database. Most fields in the boilerplate section and a summary of the I/O specifications are replicated as CMDI metadata in the repository. The user can search and view the CMDI metadata, like the metadata of any other resource. Sensitive data are not publicly visible.

A number of boilerplate fields are essential for the operation of the workflow manager: the short *ToolID*, the *PassWord*, which is used to pass maintenance responsibility to another person, the *ServiceURL*, *XMLparms*, to tell the workflow manager that the tool cannot handle simple HTTP parameters but requires parameters in XML-format, *PostData,* telling whether requests should be sent using the GET or POST HTTP method, and *Inactive*, telling whether the tool is off line at the moment.

There can be several I/O specification sections for each boilerplate section, and within an I/O specification section it is also possible to enter several alternative I/O feature specifications. The need for different sections arises in those cases where a choice of one feature has an influence on what can be chosen for another feature. An example is a tool that can take part-of-speech tags as an extra input, but only if the language is Danish or English. In that case there would be two sections, one for Danish and English saying that part-of-speech tags can be taken as input as well, and another section for all other languages, where part-of-speech tags are not mentioned as an option. In other

frameworks one may have to register each possible combination of specifications as separate profiles, which can be a lot of work if a tool supports e.g. ten languages, two file formats and three ways of presenting the output, which by multiplication result in tens of profiles.

The workflow algorithms treat all features on a par; there is no 'most important' feature. If a feature is not relevant for a tool, it should not be specified. In the current system we have good experiences with the following features: *language*, *file format*, *facet* and *presentation*. The *facet* feature describes the type of content, e.g. whether data is text, a part-of-speech annotation of a text, or gestures occurring in a video. The *presentation* feature is to humans what *file format* is to the computer. With it, we want to express that for example the tokens in a text can be presented in the 'normal' way ('running text'), or as lists sorted alphabetically or according to frequency. The same choice of presentation can be made for several other facets, such as part of speech tags and lemmas, so *presentation* and *facet* are truly orthogonal features.

The last example highlights the working hypothesis that features are and must be orthogonal and vice versa, that characteristics that can be combined rather freely are indicative of the existence of several features. As another example, a resource with *facet* feature *text* can be expressed in several *file formats*, such as an *image* of a page of a book, a *flat* sequence of characters or an *audio* file with spoken words. And from an abstract stance, one could say that some texts exist in several *languages*.

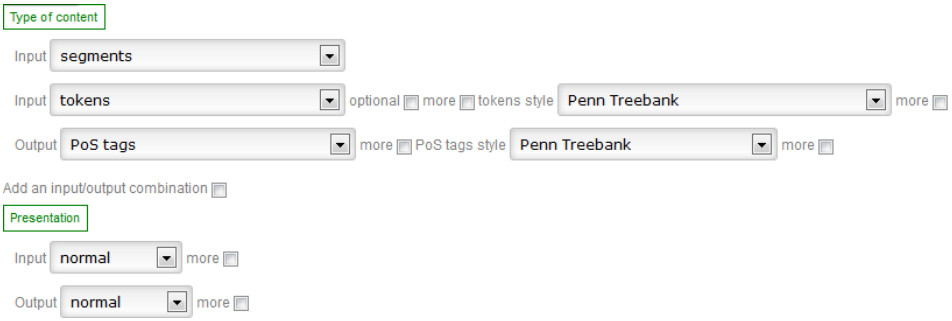

FIGURE 3: Part of registration form, showing fields for *facet* and *presentation* features. The values *tokens* and *PoS-tags* are subspecified as *Penn Treebank*. Where needed fields can be added by checking the square check boxes.

Besides values for each of these features, a tool provider in some cases also is offered the possibility to further specify a feature, in the same way as MIME types[3] consist of a media type and a media subtype. For example, once the value *image* is chosen for the feature *format*, one can choose image subspecies like *JPEG*, *TIFF*, *GIF*, etc. Here the ruling idea is that subspecifications belong to the realm of technical details that we don't want to disturb the user with unnecessarily. This is in keeping with everyday software. For example, image viewers handle wide ranges of image formats but do not

[3] http://www.iana.org/assignments/media-types

require that the user knows or even is aware of these formats. Also NLP-software may be able to read and write data in a range of formats. Nonetheless, if it is known that a tool is restricted to or has a preference for a narrow set of formats, it is possible to enumerate these during tool registration. This information helps the workflow manager getting around I/O mismatches between tools. See Fig. 3.

It is clear that devising the lists of features and feature subspecies is more of an art than science, and it is also clear that such lists will evolve in different directions in different communities, if no co-ordination is done. We have chosen to let the lists grow and evolve as new tools pose requirements that are not expressible with the current values. For interoperability between infrastructures, when adding new features we always attempt to reuse existing terminology.

The possible values for each of the four features *format*, *language*, *facet* and *presentation* are as follows.

The *format* feature can take the values *Anvil*, *audio*, *CSV*, *flat*, *HTML*, *image*, *PDF*, *RDF*, *RTF*, *TEIP5*, *TEIP5DKCLARIN*, *TEIP5DKCLARIN_ANNOTATION*, *XML*, or *video*.

The *language* feature can take 50 values. Only one of these, *Xhosa* (*xh*) is not explicitly supported by any currently integrated tool.

The *facet* feature can take the values *anonymized named entities*, *head movements*, *keywords*, *lemmas*, *lexicon*, *multiple word terms*, *N-gram frequencies*, *named entities*, *noun phrases*, *paragraphs*, *PoS tags*, *repeated phrases*, *segments*, *tagged terms*, *text*, *tokens*, and *morphemes*. This list represents the characteristics offered by the currently integrated tools and some tools that are not integrated, but which we are considering.

The *presentation* feature can have the values *alphabetic list*, *frequency list* and the intentionally vague '*normal*'.

Some combinations of feature values may seem far-fetched and can cast doubt on the tenability of the whole idea of orthogonality of features. An example is the combination of the *video* format and the *N-gram frequencies* facet. On the other hand, this may be an acceptable characterization of a video resource that lists short sequences of hand movements, sorted by frequency. Such a resource may be useful in the study of e.g. sign language. That is not to say that *all* combinations must be meaningful, but the great majority of combinations should make sense.

Currently twelve tools are installed as web services on several servers and made integral part of the infrastructure. Some tools have narrow sets of specifications, whereas others would expand into hundreds of profiles, if registering alternative I/O specifications for the same tool hadn't been allowed. The tools cover a range of formats, languages and facets.

1) *Cuneiform*[4], OCR for 23 languages,

2) *RTFreader*, a basically language insensitive program that reads *rtf* or *flat* text and writes *segments*, optionally *tokenised*, in *flat* text format,

[4] http://cognitiveforms.com/ru/products_and_services/Cuneiform.html

3) *Flat2cbf*, a program that converts *flat* text into an implementation of *TEIP5*, the CLARIN-DK basis text format *TEIP5DKCLARIN*.

4) - 7) *tokenisers* and *segmenters* for Danish and for English, taking *TEIP5DKCLARIN*-formatted text as input and producing stand-off annotations in another *TEIP5* implementation, the *TEIP5DKCLARIN_ANNOTATION* format.

8) *Brill's POS-tagger* for Danish and English,

9) *CST's lemmatiser* for 10 languages, also capable of producing a sorted list of lemmas

10) *espeak*, a very basic TTS-system for 42 languages,

11) a utility bundling *tokens*, *lemmas* and *part of speech* tags into *CoNLL-X* format, and

12) Bohnets parser[5], a syntactic dependency parser, currently for Danish only.

5 Computation of workflows

The computation of a workflow is a recursive process that starts from the user's goal and that works towards the input specification. Given a goal, the algorithm checks whether it is compatible with what is known about the input. If that is the case, a workflow solution is found. If the goal is not compatible with the know input features, the algorithm finds all tools that produce output that match the goal's features, also taking subspecifications of features into account. The input requirements of these tools are new goals, each of which is analysed in a recursive manner. See Fig. 4 for an example where two workflows were generated that both can satisfy the goal as specified in Fig. 1

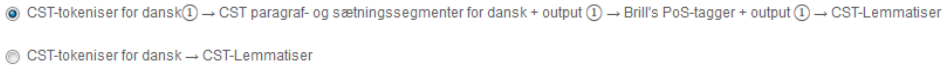

FIGURE 4: The workflows that satisfy the goal that is set in Fig. 1. By clicking a radio button, the user can choose between a tool chain of a tokeniser, a segmenter, a PoS-tagger and a lemmatiser, or a tool chain consisting of a tokeniser and a lemmatiser.

We let the recursion halt at an arbitrarily chosen maximum recursion depth of 20, which means that we do not want to ever see workflows with more than 20 tools in a chain. Once an unbroken chain of tools connects the input with the output, any features that are not specified in the goal, but specified in the input or in any of the intermediate stages, are percolated through the chain. For example, if the goal does not state the language, but it is known that the input's language feature has the value *French*, then this value is percolated until it arrives at the goal or until it stumbles into a tool that for example translates from French to Danish, in which case the value *Danish* would percolate further in the direction of the goal. Any parameters (features or subspecifications of features) that are still unresolved hereafter are not decidable for the workflow manager. In such cases the workflow manager constructs *all* fully specified workflows that are compatible with the underspecified workflow and the user is given the choice between these alternatives. If the user did not specify all four goal features,

[5] http://code.google.com/p/mate-tools/

she can normally reduce the number of generated workflow candidates by specifying more features. However, workflows can contain parameters that cannot be resolved by refining the goal specification. Such parameters must either be resolved by the user or by a heuristic implemented in the software. This is not yet done in the current system. Therefore we see that goals, especially those that need many steps, sometimes generate pages full of workflows.

6 Execution of workflows

The workflow manager functions as the hub in the execution of a workflow. The workflow manager starts with sending a request to the web service that takes the initial input. Eventually, the web service returns its result, or, if something went wrong, an HTTP error status code. In the first case, the workflow manager inspects a list of pending workflow steps, picking out and executing those that can be executed given the returned data. When there are no more pending workflow steps, the workflow manager creates a report of the workflow process, step by step naming all tools and the inputs and outputs the tools produced. Also, metadata is constructed for each result, describing how the data was created from earlier results or from the input. These provenance data, together with the intermediary and final results, is compressed in a zip-archive and stored for a limited time, currently a few days. The user receives an email with a link to a web page. When clicking the link, the web page opens in the user's browser. The user will see the report and a download link to the zip file.

For data security reasons, the user is given only one opportunity to download the zip file, which is deleted from the server as soon as it has been fetched by someone. We have chosen this rigid strategy to make it difficult to intercept the data undetected. If there is an eavesdropper on the line who fetches the results before the user tries to do that, the user will notice that something went wrong and likely ring an alarm.

After inspecting the results, the user has the opportunity to deposit the result in the clarin.dk repository, together with the intermediary results, provided that they are of a type that can be deposited. Before depositing results, the user must check and complete the metadata, because the automatically created metadata necessarily lack background knowledge that only the user can provide. The depositing of results is taken care of by a service dedicated to the depositing of resources in general, and not the workflow manager itself.

The workflow manager can asynchronously send requests to several web services at the same time, provided that the web services return immediately with an HTTP status code *202 Accepted* and send results later.

If a web service returns another status code than *200 OK* or *202 Accepted*, the remainder of the workflow is aborted and the user receives an email telling where the workflow got off track. The user still receives the results that were created successfully.

7 Limitations and solutions

The chosen approach has its problems and disadvantages, like any robotic system that tries to replace a human expert.

Whereas user-friendliness and accepted standards are good organizing forces, there is also a bad, but at times inescapable, disorganizing force. Sometimes, when we try to register a new tool, we may discover that a subspecification of a feature value itself needs further specification. But since the architecture only allows two levels of specification, we are forced to heighten the status of the subspecification to that of a feature value, and to introduce the further specifications as subspecifications of the new feature value. If this causes too much terminological pain and exposes too much technical detail to the user, we may have to reconsider the integration of the tool that is in need of the extra distinction. Is the tool really mature, or should it be adapted to accept a wider range of inputs before we attempt to integrate it in the infrastructure? It must be added that changing the existing terminology should not be done light-heartedly, because not only will some tools have to be re-registered, the web services that wrap around these tools will have to be fixed as well, because they will receive parameters with altered names.

On a more theoretical level, we may question whether all goals can be stated in only a few (1-4) words. However, adding even a single extra field to supplement the goal description can easily be more confounding than of help, because after the features *language* and *file format* it is very hard to define generic features that feel natural for users.

Because workflows are created on the fly, they do not have persistent metadata and identity of their own. Yet, together with the output, the user receives a full specification of each step in the workflow, so the same workflow can be chosen the next time the user wants to do a similar task. This workflow documentation only partly compensates for the lack of persistent workflows. Therefore we have plans to make a resource type 'Workflow' that can be stored in the repository and reused later. This will make it easier to repeat a workflow at a later time.

Certain tools and kinds of input do not easily fit in the chosen scheme. Tools that require user interaction cannot be integrated, because the execution of workflows is in batch mode. Neither can tools that output metadata rather than data, such as language guessers, be incorporated in our workflows, because they would constitute decision points in workflows that cannot be computed on beforehand. As mentioned before, the automatically generated workflows are linear sequences of instructions, without the possibility to react to conditions that arise at run time, such as taking one or the other workflow branch depending on the outcome of a language guesser.

8 Conclusion and Outlook

The clarin.dk workflow manager is fully and reliably functioning, and response times of the user interface are very fast. We already have a modest number of tools, some of which are very versatile and can be seen as Swiss Army knifes among NLP tools. We have concrete plans to add more tools and also welcome contributions from other tool providers.

The chosen approach cannot completely replace workflow strategies as implemented in full scale workflow editors, because workflows created by the clarin.dk workflow manager are chains without branching points that depend on the results of earlier steps,

but the workflows that can be realised, are made available in a user friendly manner, not requiring expert knowledge and making optimal use of the available tools.

From a programmer's point of view, it is very rewarding to see that an accurate description of a tool is enough to see it pop up as a step in a workflow, and it is even more rewarding to see that it works. We think that tool providers appreciate this interaction, and that they invest time in improvements to their tools to make them more general, robust and of higher quality.

Acknowledgments

DIGHUMLAB Digital Humanities Lab Denmark is supported by Danish Agency for Science, Technology and Innovation, Ministry of Science, Innovation and Higher Education.

The preparatory project DK-CLARIN was also supported by the Danish Agency for Science, Technology and Innovation, as well as by all eight partner institutions. We want to thank all partners for their contribution.

References

Cristea, D., Pistol, I. (2008): Managing Language Resources and Tools Using a Hierarchy of Annotation Schemas. In *Proceedings of the Workshop on Sustainability of Language Resources*, LREC-2008, Marrakech.

Funk, A., Bel, N., Bel, S., Büchler, M., Cristea, D., Fritzinger, F., Hinrichs, E., Hinrichs, M., Ion, R., Kemps-Snijders, M., Panchenko, Y., Schmid, H., Wittenburg, P., Quasthoff, U. and Zastrow, T. (2010): *Requirements Specification Web Services and Workflow Systems*. Available at: http://www-sk.let.uu.nl/u/D2R-6b.pdf

Hinrichs, E., Hinrichs, M., Zastrow, T. (2010) WebLicht: web-based LRT services for German. In *ACLDemos '10 Proceedings of the ACL 2010 System Demonstrations*, Pages 25-29, Association for Computational Linguistics Stroudsburg, PA, USA.

Kemps-Snijders, M., Brouwer, M., Kunst, J. P. and Visser, T. (2012): Dynamic web service deployment in a cloud environment. In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, Istanbul, Turkey, May 23-25, 2012: 2941-2944, European Language Resources Association (ELRA)

Offersgaard, L. Jongejan, B. and Maegaard, B. (2011). How Danish users tried to answer the unaskable during implementation of clarin.dk. In *SDH 2011 – Supporting Digital Humanities*, Copenhagen.

Offersgaard, L., Jongejan, B., Seaton, M. and Haltrup Hansen, D. (2013). CLARIN DK – status and challenges. In *Proceedings of the Nordic Language Research Infrastructure Workshop at NoDaLiDa, Oslo, May 22, 2013*