Handling Big Data and Sensitive Data Using EUDAT's Generic Execution Framework and the WebLicht Workflow Engine.

Claus Zinn, Wei Qui, Marie Hinrichs, Emanuel Dima, and Alexandr Chernov
University of Tübingen
Wilhelmstrasse 19, 72074 Tübingen, Germany
{claus.zinn, wei.qui, marie.hinrichs, emanuel.dima, alexandr.chernov}@sfs.uni-tuebingen.de

Abstract

Web-based tools and workflow engines can often not be applied to data with restrictive property rights and to big data. In both cases, it is better to move the tools to the data rather than having the data travel to the tools. In this paper, we report on the progress to bring together the CLARIN-based WebLicht workflow engine with the EUDAT-based Generic Execution Framework to address this issue.

Keywords: Generic Execution Framework, WebLicht, Big Data, Sensitive Data

1. Introduction

Our work addresses two challenges that affect the applicability of workflows for some data sets. First, restrictive property rights may forbid research data to leave their home institution, and therefore, data transferal to other institutions for processing is not allowed. The second issue concerns the size of the data, with big data often causing prohibitive overhead once it is necessary to send such data back and forth to the various tools of a scientific tool pipeline. In both cases, it is desirable to bring the workflow engine to the data, rather than having the data travel to the tools.

The EUDAT project is currently developing the Generic Execution Framework (GEF). The GEF aims at providing a framework that allows the execution of scientific workflows in a computing environment close to the data. In this paper, we describe how WebLicht needs to be adapted to render its services compatible with the GEF.

The remainder of the paper is structured as follows. In Sect. 2, we give some technical background on the Generic Execution Framework and WebLicht. Sect. 3 specifies the GEF-WebLicht integration from WebLicht's side. This includes user-centric front-end considerations as well as necessary adaptations to WebLicht's back-end and the processing services connected to WebLicht, which need to be converted to GEF compatible web services. An integral part is a translation mechanism that liaises the WebLicht workflow engine with the GEF environment, and which ensures that all data transfers occur at the GEF's host institution, which also hosts all data. Sect. 4 reviews the service compatibility of WebLicht from the point of view of a GEF administrator, and in Sect. 5, we reconsider WebLicht's perspective. In Sect. 6, we conclude.

2. Background

2.1. The Generic Execution Framework

One underlying motivation for the GEF is that datasets have become much larger than the tools that process them. It is thus more efficient to move the tools to the data rather than the data to the tools, given that the tools do not require a large amount of processing power. This argument is often supported by restrictive property or privacy rights attached to the data; here data transfer to tools not under control by the property holder is often not possible. Also, the transfer of big data sets across the tools of a tool chain often leads to bottlenecks, and transfer time is sometimes higher than the actual processing time.

The GEF aims at a framework that allows developers to pack tools (and their computation) into movable containers that can be moved towards the data. GEF makes use of Docker [U1], a virtualization mechanism that wraps an application into a container that capsules and sandboxes all computation, that is, everything needed to run the application, including all system tools, system libraries, and settings. Docker-based technology shows a very good performance, with container images that can have a small image size, and are fast to start. Container images are immutable by design and relatively easy to specify, using Docker scripts and pre-built container images to build-upon.

Fig. 1 shows the design rationale of GEF (Dima et al., 2015). Docker software runs on the operating system of the host, on the host's hardware, and is able to execute Docker containers (holding the tools' virtualization). The

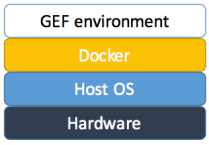

Figure 1: The GEF architecture.

GEF software is built-upon Docker allowing community admin users to upload their tools together with a Dockerfile from which Docker container images are generated. These images become GEF services and can be invoked on any data set in the EUDAT Collaborative Data Infrastructure (EUDAT, 2017), see [U2].

The GEF source code is available at GitHub [U3], and there is available a 1.0.0-beta release. The software's back-end is built upon the Go programming language [U4], and its front-end is built upon the React Javascript library for building user interfaces [U5]. Interested parties can use the software, as a GEF environment has been set-up for public use, see [U8].

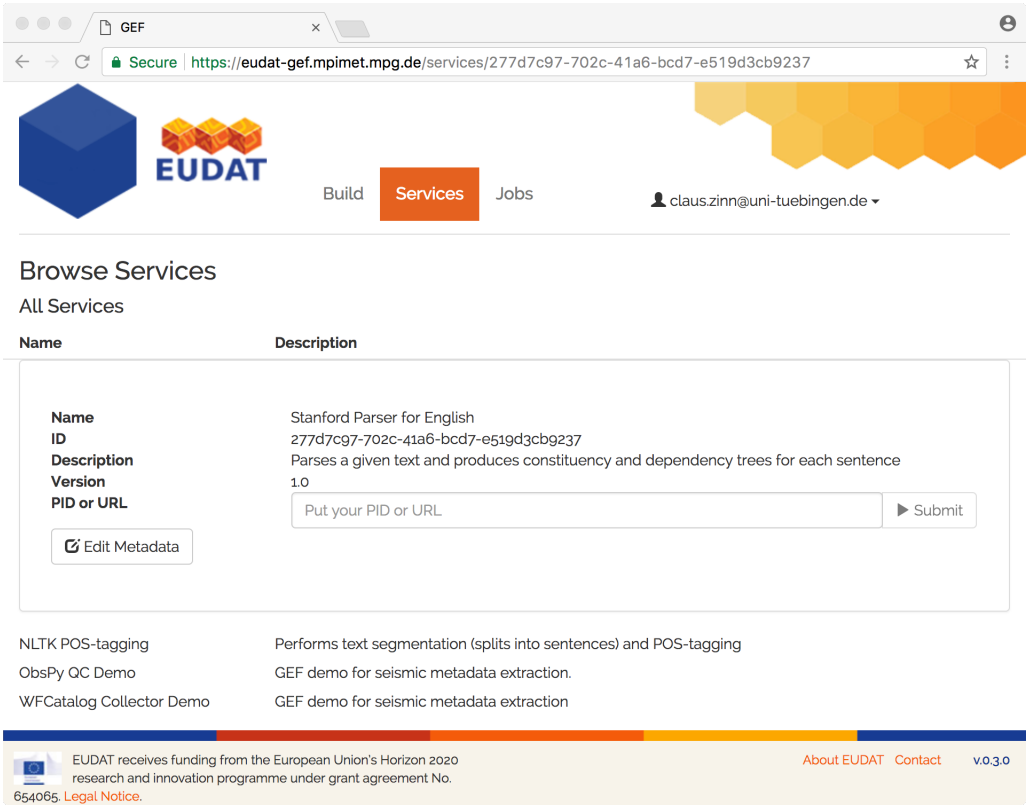

Figure 2: Main Page of the GEF front-end, see [U8].

Fig. 2 shows the front-end of the GEF. In the top pane, there are three items: "Build", "Services" and "Jobs". When choosing the first item, the GEF admin ("power user") can build a new GEF service. For this, the user needs to upload a Dockerfile, together with other files which should be part of the container. With the file upload complete, GEF's underlying Docker software will build a Docker-based container image, which the GEF will use to provide the corresponding GEF service of the application.

Fig. 2 shows four services from the GEF test site. A GEF service can be run by supplying a PID or URL that points to the input to be processed. Hitting the "Submit" button, will start the service asynchronously. All running jobs can be inspected through "Jobs". Once a job terminates, its output volume holds the result of the computation.

2.2. The WebLicht Workflow Engine

WebLicht is a workflow engine giving users a web-based access to over fifty tools for analysing digital texts (Hinrichs et al., 2010). Its pipelining engine offers predefined workflows ("Easy Mode") and supports users in configuring their own ("Advanced Mode"). With WebLicht, users can analyse texts at different levels such as morphology analysis, tokenization, part-of-speech tagging, lemmatization, dependency parsing, constituent parsing, co-occurrence analysis and word frequency analysis, supporting mainly German, English, and Dutch. Note that WebLicht does not implement any of the tools itself but mediates their use via pre-defined as well as user-configurable process pipelines. These workflows schedule the succession of tools so that one tool is called after another to achieve a given task, say, named entity recognition.

WebLicht is a good step forward in increasing (web-based) tool access and usability as its TCF format mediates between the various input and output formats the tools require; and it calls the tools (hosted on many different servers located nation and world-wide) without any user engagement. WebLicht has now been used for many years in the linguistics community, and it is the workflow engine of choice for many national and European researchers in the CLARIN context. WebLicht is actively maintained, profits from regular tool updates and new tool integrations, and has recently been integrated with TüNDRA, a treebank search and visualization tool that allows WebLicht users to inspect linguistically annotated data (Chernov et al., 2017). With TüNDRA, users can search for specific linguistic phenomena at the word and sentence level, and visualize such phenomena. The WebLicht workflow engine is closed source, but all members of the CLARIN Service Provider Federation can access its services [U6]. WebLicht-related technology is maintained in a public Github repository [U7].

Workflow definition and execution. Each tool in the workflow is identified by a persistent identifier that WebLicht's execution system uses to invoke the tool. With each tool invocation, WebLicht passes on a TCF-compliant data file that contains the tool's input. The tool processes selected parts of the input, and enriches the TCF file with the output of the computation, which is then sent back to the WebLicht orchestrator. WebLicht then passes the enriched TCF file to the next tool of the workflow until all tools in

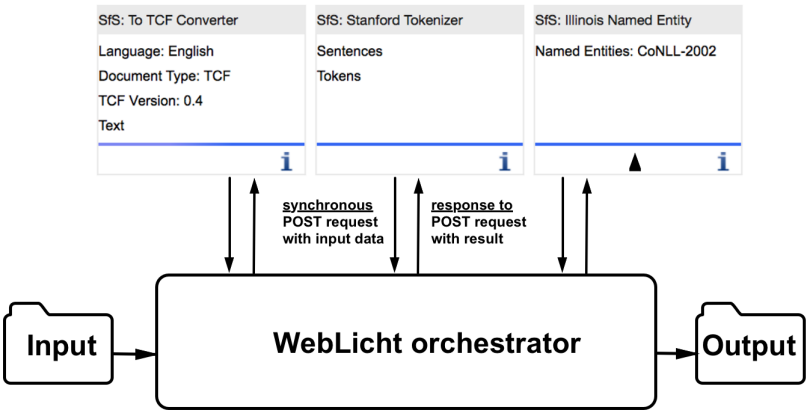

Figure 3: Data Flow in WebLicht.

the workflow have been executed in the given order.

Fig. 3 shows the data flow between the WebLicht orchestrator and the tools. Note that each tool is invoked via the Hypertext Transfer Protocol (HTTP) using a POST method, with the tools' output data being captured in the response body of the request. This design decision makes it hard to use WebLicht on huge data sets, or on data that cannot leave the host institution for property rights or privacy concerns.

In the GEF mind set, WebLicht's tools need to be "migrated" to a GEF environment that encapsulates all computation. To keep all data transfer local to the GEF environment that also hosts the data, the way the WebLicht orchestrator invokes the services needs to be changed.

3. Integration of WebLicht into GEF

GEF is not capable of executing workflows by itself. To execute a GEF-based workflow, it must be orchestrated by an external engine, which makes reference to a GEF-based repository of (immutable) application containers, each of which can be referenced by a PID. In the sequel, we will describe how the WebLicht orchestrator can be used to execute a GEF-based WebLicht workflow; we also describe how any data transfer between the GEF-external WebLicht orchestrator and the GEF-internal services is handled.

3.1. WebLicht Adaptations

WebLicht Front-End. Fig. 4 depicts a proposed UI change to WebLicht's entry page. In addition to the "Input Selection" pane, there is a "Tool Location" pane, where users can select the tool environment that the WebLicht orchestrator should use. The environment "GLOBAL – all tools" is the default tool environment where WebLicht assumes traditional operation (all tools are eligible for inclusion in the workflow). When the user selects the environment "GEF@sfs_tuebingen", the WebLicht orchestrator will only make use of workflows whose tools are running in the GEF environment at the SfS in Tübingen. In GEF mode, no data is uploaded to WebLicht, rather users specify a data URL that points to the data, usually located close to the chosen GEF computing environment.

WebLicht Back-End. The backend adaptations to WebLicht are more substantial. Two changes to WebLicht's back-end are discussed: how should the orchestrator organise its tool space; and how should the orchestrator call the individual tools to minimize data transfers?

The WebLicht orchestrator uses the tools' metadata to help users define workflows. For this, WebLicht is harvesting tool metadata from tool providers (as ingested to the metadata repositories of the CLARIN centre registries). We assume that providers of GEF-ified tools advertise them in the same way so that WebLicht is aware of them. The description of a WebLicht tool is based on the CMDI Profile `WebLichtServiceProfile`, which is addressed via a persistent identifier in the tool's metadata. Among other information, the metadata provides information about the tool's invocation URL and its input and output parameters. By default, the WebLicht Orchestrator works on the entire tool space that results from the tools' metadata harvest. When a WebLicht user specifies a GEF environment of her choice, the orchestrator must cut through this space. In Easy Mode, it needs to identify which predefined workflows are available in the GEF organisation specified by the user (a workflow is available in a GEF environment if each tool of the workflow is available in the GEF environment). In Advanced Mode, the Weblicht orchestrator ignores all tools not originating in the given GEF environment; only tools that are part of a given GEF environment are considered for workflow construction.

Fig. 3 shows the back-and-forth movement of data between the WebLicht orchestrator and the tools it is executing. If a given workflow consists of n tools, then the data is transferred 2 times n between the orchestrator and the tools, potentially across networks of varying bandwidth and latency. With data required to stay in the GEF environment, the interplay between the orchestrator and the tools must change; rather than passing along actual data, now references to data originating in the GEF environment are being passed.

This requires the "wrapping" of tools. A wrapped tool: (i) retrieves the input data from a given URL, (ii) starts the tool it wraps by posting the tool the input data via HTTP, (iii) accepts the response of the HTTP request, (iv) stores

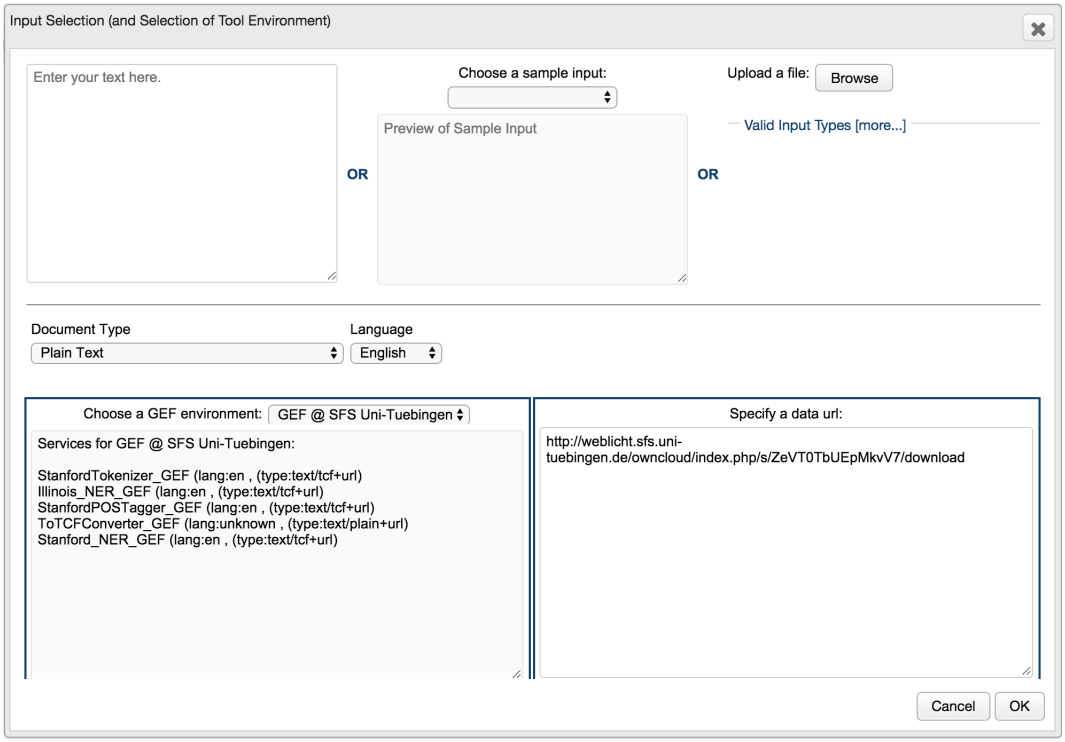

Figure 4: Screenshot of WebLicht GEF extension.

the output data locally (no transfer), and (v) returns this location with a corresponding URL. With the workflow consisting only of tools with such wrappers no data is sent back and forth to WebLicht, but rather URL-based pointers to the data. Note that the wrappers will be required to monitor the execution of the tools they wrap. That is, when an HTTP POST request yields an error message, then this message is sent back to the WebLicht orchestrator, which then prematurely terminates the execution of the workflow with an appropriate error message. Once, a tool is wrapped, it needs to get Dockerized and ingested into the GEF.

3.2. Liaising between WebLicht and the GEF

A translation mechanism is needed to liaise between WebLicht and the GEF-enabled services in a GEF environment. Consider a user wanting to annotate sensitive data, say an English text with named entities; for this, the user selects a GEF-based WebLicht workflow, whose corresponding XML representation is given in Fig. 5.

```
<cmd:Toolchain>
    <cmd:ToolInChain>
        <cmd:PID>http://hdl.handle.net/11858/00-1778-GEF-0004-BA56-7</cmd:PID>
        <cmd:Parameter value="en" name="lang"/>
        <cmd:Parameter value="text/plain" name="type"/>
    </cmd:ToolInChain>
    <cmd:ToolInChain>
        <cmd:PID>http://hdl.handle.net/11022/0000-GEF-2518-C</cmd:PID>
    </cmd:ToolInChain>
    <cmd:ToolInChain>
        <cmd:PID>http://hdl.handle.net/11022/0000-GEF-839F-9</cmd:PID>
    </cmd:ToolInChain>
</cmd:Toolchain>
```

Figure 5: The NER EasyChain in XML (fragment).

Each of the three tools in the tool chain is referenced with a persistent URL, pointing to metadata that describes the tool, see for instance, Fig. 6.

Stanford Tokenizer Stanford Tokenizer is a an efficient, fast, deterministic tokenizer.

Stanford Tokenizer is a an efficient, fast, deterministic tokenizer.

PID	http://hdl.handle.net/11022/0000-0000-2518-C		
URL	http://bridgit.sfs.uni-tuebingen.de/StanfordTokenizer		
Email	wlsupport@sfs.uni-tuebingen.de	Status	production
Created	2014-07-07T11:45:58.789+02:00	Modified	2014-07-07
Input		Output	
type	text/tcf+ url	sentences	
version	0.4	tokens	
text			
lang	en		

Figure 6: Metadata for the Stanford Tokenizer..

The metadata has a URL slot that is used to invoke the tool from WebLicht. Note that the URL does not point to the GEF environment, but rather to the translation mechanism, see below. Also note that the type of the input is not "text/tcf+xml" but "text/tcf+url", indicating that WebLicht will need to invoke the tool by passing a reference to the data to the tool, rather than the actual data.

Fig. 7 summarises the interaction between the WebLicht orchestrator and the GEF environment. As with all other web services connected to WebLicht, the orchestrator invokes the given URL with a synchronous POST request. Given the new type "text/tcf+url", no input data is being passed, only a URI reference to the data. The synchronous request invokes BridgIt, the translation mechanism, which liaises with the GEF environment. BridgIt keeps a record of:

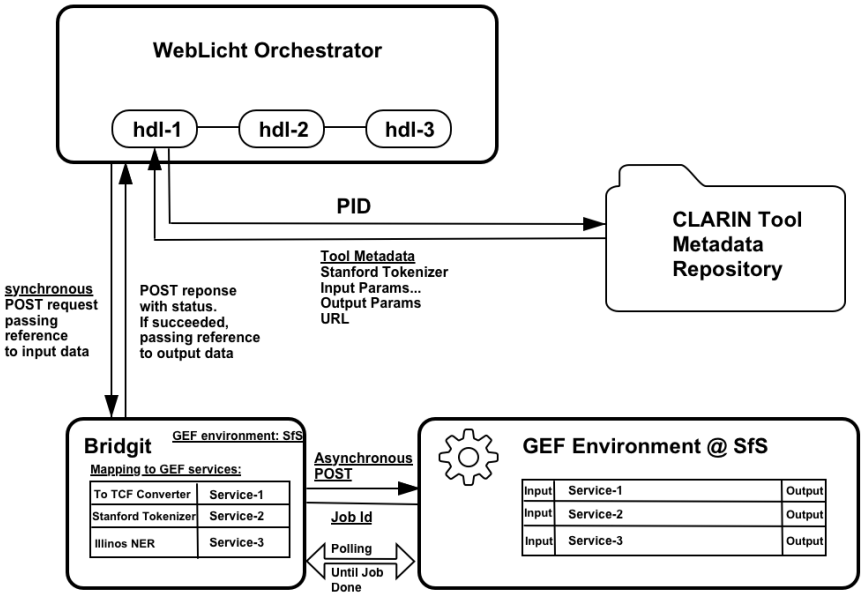

Figure 7: Architecture of WebLicht - GEF interaction.

- a pointer to the GEF environment, say, `http://gef.sfs.uni-tuebingen.de:8443`.
- a JSON-based table that maps URL-encoded information such as "/StanfordTokenizer" to a service code that identifies the corresponding GEF service.

With this information the URL `http://gef.sfs.uni-tuebingen.de:4041/jobs?serviceId=<id>&pid=<pointerToData>` is constructed and BridgIt sends this URL as POST request to start the GEF service. The given GEF environment answers with a JSON structure that gives the job identifier of the corresponding GEF service. Bridgit then polls the GEF environment at regular intervals (200 ms) to check whether the job has terminated. This is a GET request of the form `http://gef.sfs.uni-tuebingen.de:4041/jobs/<jobId>`. This GET request returns all information about the GEF job with the given jobID: the job's id, state, and output volume (for the result of the processing). When the job terminated successfully, then BridgIt returns a reference to the output volume associated with the job id; otherwise BridgIt passes on the error code encoded in the response of the GET request. Any result is returned to WebLicht as response to the synchronous POST request to BridgIt.

4. GEF Perspective

Setting-up a GEF environment comes at a substantial cost. Technical and logistical support is needed to keep costs manageable. From the perspective of a GEF community manager, it is highly desirable that all tools he or she wants to integrate into a GEF environment are pre-packaged. The dockerization of a tool requires intricate knowledge about the tool and its best runtime environment. In general, such information is rarely available. As a consequence, we assume a GEF community manager to contact the WebLicht developers and asks them for support. The step from a Dockerized web service and the GEF-wrapper is small.

Setting-up a GEF environment must also include the provision of metadata records that describe all GEF-ified tools using the CMDI profile `WebLichtWebService`. The metadata must be included in the GEF host's CLARIN centre repository so that WebLicht can harvest this data, and hence knows about the existence of the GEF environment. For this, the existing metadata of the original web service should be adapted accordingly.

The GEF community manager must also complement the GEF environment with the translation mechanism. Here, we assume that a reference implementation of BridgIt will be provided by the WebLicht team so that a GEF admin can use and configure the translation mechanism easily.

Given the required expertise and high cost of setting-up a GEF environment, the WebLicht team may provide GEF environments for popular natural language processing chains. Once a GEF environment has been set-up, the cost of moving it to a new location (close to the data) should come at relatively cheap cost.

5. The WebLicht Perspective

The standard, browser-based use of WebLicht [U6] does not handle big data well; it imposes file size limits and the processing is threatened by session timeouts. For files larger than 3 MB, users are advised to us WebLicht as a Service (WaaS), which is a REST service that executes WebLicht chains [U9]. WaaS allows users to run WebLicht chains from their UNIX shell, scripts, or programs, and hence, users are not exposed to browser-based timeouts, which vary across browsers. Note however, that chain

execution in WaaS produces the same data flow than the browser-based version, see Fig. 3. That is, data is sent back and forth to all web services in the chain via HTTP post requests, which is expensive. The work described in this paper improves the situation for larger files as the WebLicht orchestator now invokes GEFified tools by sending them references to the data rather then the actual data. But there is ample potential for optimization. The wrapping of services described earlier introduces an unnecessary overhead. Here, it is advisable to Dockerize the initial tools (that is, not their REST-based variants), and have them accessing their input via mounted Docker volumes (instead of responding to HTTP requests).

Another path for optimization is WebLicht's TCF format, which is rather verbose when compared with other competing formats (Lapponi et al., 2014). As a matter of fact, each tool in a WebLicht workflow enriches the TCF input with its processing result. Reconsider the easy-chain displayed in Fig. 3. Here, the Stanford Tokenizer tokenizes all input as preprocessed by the TCF converter, and then adds all tokens to the TCF file; this stage hence more than doubles the data, which is then sent to the Illinois Named Entity Recognizer. Assuming that the Illionois NER relies on the tokens only, the original input could have been consumed by the Stanford Tokenizer, nearly halving the data then sent to the next stage. Here, WebLicht's original design rationale is to blame. It regards all web services connected to WebLicht as black boxes and abstracts from their inner workings. For big data, this abstraction may prove too wasteful. Future tool metadata descriptions might need to specify whether a tool, say, requires only tokenized input or whether they also need access to the original input. WebLicht's workflow engine would then invoke the services with a 'consume' flag set to true or false, respectively.

In addition, the TCF format itself could be compacted in many different ways without inflicting any information loss. A JSON-based representation of TCF, for instance, promises to cut space requirements by half.

In the paper, WebLicht breaks out of its current boundaries. It empowers users to not only select the tools for their workflow, it lets them now also choose where the tools should run (see the selection of the GEF environment in Fig. 4). Here, more flexibility can be added by allowing *mixed* workflows: here, the first stages of the workflow, for instance, must be processed by in-house (*i.e.*, GEFified) tools, while the latter processing stages could be processed by non-GEF services (assuming that the original input is 'consumed' by the prior stages, see above).

To use WebLicht and its associated tools, users need to be authenticated with an account of the CLARIN Service Provider Federation. For users to process sensitive data with WebLicht, a future version will need to ask users for additional credentials. For this, reconsider, Fig. 4. When users specify a data url for the input data (bottom right), WebLicht could mediate users' credentials with the data site to check whether a given user is authorized to access the data. Only after successful AAI, users will be delegated to the next GUI page to identify or construct workflows.

6. Discussion and Conclusion

We are in the process of setting-up a proof of concept GEF environment for a popular language processing workflow (Named Entity Recognition for English). So far, we have dockerized all three tools of this workflow (including their service wrapping). This is the essential step towards "movable computation". The BridgIT liaison device is currently being developed and tested, and CMDI-based metadata descriptions for GEF-enabled services are being authored and made available for harvest by WebLicht. For testing purposes, we will create a WebLicht branch that offers GEF-enabled services to users.

Our work has already had a positive impact on the development of both WebLicht and GEF. To limit the amount of data transfer between the original WebLicht orchestrator and the individual services that define a workflow, it is now being discussed whether services shall also be invokable by passing references to the data rather than by posting the data to them. Our work also informed the EUDAT-GEF development team to ensure that user requirements stemming from the WebLicht use case lead to GEF feature requests that will find their way into the official GEF specification and implementation. One feature request concerns the invocation of a GEF service. At the time of writing, starting a GEF-ified tool means starting a new Docker container. Some tools, however, consume significant resources at start-up. The Illinois Named Entity Recognizer, for instance, loads large models to inform its processing. Here, it is more advisable to not terminate the GEF service (that is, to not stop the Docker container) once NER processing finished. In this case, the GEF service should continue listening to and serving future incoming processing requests.

We have started our article with GEF's underlying motivation that datasets have become much larger than the tools that process them, or that there are datasets that are not allowed to leave their hosting institition for legal reasons. Moving the tools to the data rather than the data to the tools seems reasonable. In linguistics, there is sensitive data, but big data issues become more prominent, too. Take, for instance, the Newsreader project with the aim to parse 100.000+ news articles live on a daily basis (Vossen et al., 2016). For these projects, a future version of WebLicht based on our approach can play a key role in orchestrating and executing a variety of workflows to gather, collect and post-process such data. The integration of the CLARIN WebLicht workflow engine and its services with EUDAT's Generic Execution Framework makes it possible to bring the language processing tools to an execution environment that also hosts the data, hence allowing language processing of sensitive and big data.

7. Acknowledgements

This work has been supported by the EUDAT (grant agreement No. 654065) and CLARIN-PLUS (grant agreement no. 676529) projects, both funded from the European Union's Horizon 2020 research and innovation programme. We would like to thank the anonymous reviewers for their valuable feedback.

8. URL References

[U1] https://www.docker.com/what-docker
[U2] https://www.eudat.eu/eudat-cdi/about
[U3] https://github.com/EUDAT-GEF/GEF
[U4] https://golang.org
[U5] https://facebook.github.io/react
[U6] https://weblicht.sfs.uni-tuebingen.de
[U7] https://github.com/weblicht
[U8] https://eudat-gef.mpimet.mpg.de
[U9] https://weblicht.sfs.uni-tuebingen.de/WaaS

9. Bibliographical References

Chernov, A., Hinrichs, E., and Hinrichs, M. (2017). Search your own treebank. In *Proceedings of the 15th International Workshop on Treebanks and Linguistic Theories (TLT15)*, pages 25–34. Bloomington, IN, USA.

Dima, E., Pagé, C., and Budich, R. (2015). D7.5.2: Technology Adaptation and Development Framework (final). Technical report, EUDAT deliverable.

EUDAT. (2017). The EUDAT Collaborative Data Infrastructure (CDI). See the website at https://eudat.eu/eudat-collaborative-data-infrastructure-cdi.

Hinrichs, E., Hinrichs, M., and Zastrow:, T. (2010). Weblicht: Web-Based LRT Services for German. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (System Demonstrations)*.

Lapponi, E., Velldal, E., Oepen, S., and Knudsen, R. L. (2014). Off-road laf: Encoding and processing annotations in nlp workflows. In N. Calzolari (Conference Chair), et al., editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland, may. European Language Resources Association (ELRA).

Vossen, P., Agerri, R., Aldabe, I., Cybulska, A., van Erp, M., Fokkens, A., Laparra, E., Minard, A.-L., Aprosio, A. P., Rigau, G., Rospocher, M., and Segers, R. (2016). Newsreader: Using knowledge resources in a cross-lingual reading machine to generate more knowledge from massive streams of news. *Knowledge-Based Systems*, 110:60 – 85.