Reproducible Annotations

by

Alexander Leonhardt

A thesis presented in partial fullfillment to attain the degree
Bachelor of Science

Goethe University Frankfurt
Germany
December 28, 2021

Reproducible Annotations

by
Alexander Leonhardt

Abstract

This bachelor thesis presents a software solution which implements reproducible annotations in the context of the UIMA framework. This is achieved by creating an automated containerization of arbitrary analysis engines and annotating every analysis engine configuration in the processed CAS document. Any CAS document created by this solution is self sufficient and able to reproduce the exact environment under which it was created. A review of the state of the art software in the field of UIMA reveals that there are many implementations trying to increase reproducibility for a given application relying on UIMA but no publication trying to increase the reproducibility of UIMA itself. This thesis improves upon that technological gap and provides a throughout analysis at the end which shows a negligible overhead in memory consumption but a significant performance regression depending on the average document size and the analysis engine pipeline which was examined.

Acknowledgement

I want to express special thanks to Mr. Guiseppe Abrami who encouraged me to think out of the box and always helped me improving the existing solution. Furthermore I want to thank Mr. Daniel Baumartz who spent a lot of time together with Mr. Abrami to introduce me to the TextImager and has always taken the time to support my research and start another experiment on the server. Additional thanks go to Professor Alexander Mehler who held an excellent lecture which sparked my interest for the field of NLP and provided valuable insights on how to improve the edge cases of this thesis. Special thanks also go to my family and my girlfriend who always supported me during this time.

Contents

1. Introduction

One of the distinguishing features of anything that aspires to the name of science is the reproducibility of experimental results.35

Matthew Stewart

Reproducibility is an important part of the research. In computer science, the ability to reproduce a given experiment is determined by many factors like hardware configuration, operating system, programming environment, and many more. Recent publications have gone as far as speaking of a "Reproducibility crisis" (Baker and Penny, 2016; Belz et al., 2021) in science in general. Major conferences start to give workshops on the topic of reproducibility; LREC'2020 had a shared task on the topic of reproducibility (Branco et al., 2020), EMNLP'20 and AAAI'21 have adopted the ML reproducibility checklist(Belz et al., 2021; Whitaker, 2020) highlighting the increased conscience of researchers on the topic of reproducibility. Being able to reproduce own results and results from others is therefore an important and actively researched part of computer science. With the increasing demand for a stronger focus on reproducibility in the field of computer science arises the need for a new framework which improves the reproducibility of the *Unstructured Information Management Architecture*, UIMA, one of the most used frameworks in the field of NLP. UIMA which is already called "a de facto standard" (Abrami et al., 2020) of NLP powers different applications due to the flexibility of the dynamic type system *(TSD - Type system descriptors)* and the large amount of components released for UIMA. Specifically one main component of UIMA based annotation systems is improved, the UIMA analysis engine description. By improving upon this single component of the UIMA framework the reproducibility of most analysis engine systems can be improved even in a cross programming language concept. While UIMA supports multiple programming languages by means of Swig[1] it has mostly been a domain of the java programming language. Nevertheless with new implementations of the UIMA type system in the programming language python (Klie and de Castilho) and with the python community in general expressing an up picking interest in NLP, (Granroth-Wilding, 2020; Neumann et al., 2019; Qi et al., 2020), a multiprogramming language capable framework to facilitate reproducibility is needed.

1.1. Necessary attributes

There are several requirements for a framework that aspires to deliver reproducible annotations based on the UIMA framework. The most important requirements are listed below to provide the reader with an overview of the attributes deemed necessary. An elaboration of the scientific basis of these requirements is found in chapter 2.

(a) **Multi programming language friendly:** The uprising of python in the field of ML and also in the field of NLP makes the inclusion of multiple programming languages a necessity.

[1]http://www.swig.org/, last acessed at: 12/27/2021 22:41 MET

(b) **Operating system and environment agnostic:** The operating system and the environment of the researchers are distinct in almost every setting. Installed libraries, provided standard function implementations (pseudo-random number generator (Alzhrani and Aljaedi, 2015)), standard charset UTF-8, ASCII and many more, operating system language, and timezone settings greatly affect the outcome of operating system independent programming languages (seen in an influential glitch in computational chemistry (Bhandari Neupane et al., 2019), the script depended on the file ordering of the operating system). A promising approach to reproducibility reduces the differences between the original environment and the reproducers environment as much as possible.

(c) **Scalable:** NLP employs large machine learning models for a variety of tasks and therefore needs a lot of processing power. To improve adaptability it is mandatory to provide a convenient way of scaling the provided solution to a multi server system.

(d) **Composable:** The annotation of a document with multiple reproducible annotations or pipelines should yield a document or configuration which encompasses all used parameters and annotators so that the reproduction derived from the multiple annotations may yield the same annotations again.

(e) **Parameter and annotator discovery from data:** There needs to be a way to obtain the parameters, annotators and package versions of the classes which annotated the documents. Without this information it is impossible to recreate the annotations made on the document at a later time point. Code itself provides parameter and annotator discovery, but the data is not connected to the code at all. Code is subject to frequent changes and any change to the code can destroy the reproducibility of the previously obtained results. Alone from the data it should be possible to recreate the parameters and annotators. This tight coupling prevents the data to become detached from the code that created it.

After reviewing the literature it becomes clear that this thesis implements a unique solution by annotating the environment and used pipelines directly in the documents themselves. No other framework or publication could be found which uses UIMA to create a self sufficient reproducible document which encapsulates all pipelines used on it. Therefore the review of the literature will focus on reproducibility of state of the art implementations in general. Current solutions have either not considered a multi-programming language environment (de Castilho, 2014) or provided containerization of the whole application, GATE, or cTakes (Digan et al., 2021), which increases reproducibility in the applications but has limited scope due to the focus on the application instead of the underlying framework. The PEAR format created by UIMA to establish a common standard for sharing analysis engine components has a multitude of problems as found out by (Hahn et al., 2016). A more in depth analysis of current state of the art applications is being performed in chapter 3. Closing this technological gap induced by the lack of a general solution for UIMA based research supporting multi-programming language, operating system agnostic reproducible annotations, is the main goal of this thesis.

The next paragraph shall give an overview of the used definition of the term reproducibility. This is needed due to the differing definitions in scientific literature about the term reproducibility and replicability (Drummond and Drummond, 2009; Belz et al., 2021; Rougier et al., 2017). In this thesis the definition of the term leans on (Belz et al., 2021) definition of reproducibility who decided after a systematic review to use the International Vocabulary of Metrology VIM to define the term in a concise way. Mapping these definitions to the NLP context yields the two different types of reproducibility which will be used in this thesis:

1. **Reproducible annotations on the same system:** Recreating the annotations on the same system with almost exactly same results (only differing in timestamps and IDs). Leaning on (Belz et al., 2021) *reproducibility under same conditions.*
2. **Reproducible annotations on a different system:** Reproducibility achieved when changing the operating system, environment, language, time zone or character encoding. Leaning on (Belz et al., 2021) *reproducibility under varied conditions.*

The thesis will focus on the second type of reproducible annotations which can be seen as a super set to the first type of reproducible annotations this can be seen in fig. 1.1.

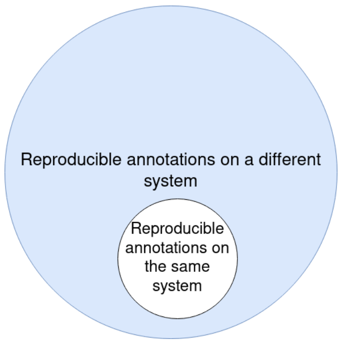

Reproducible annotations on a different system
Reproducible annotations on the same system

Figure 1.1.: The set of reproducible annotations on a different system is a superset to reproducible annotations on the same system: $R_{same} \subseteq R_{different}$.

The implementation *DockerWrapper* introduces a scalable way of wrapping an UIMA analysis engine. An *DockerWrapper* itself is an UIMA annotator and by this can be used to build an analysis engine which then executes itself in a docker container. Every CAS document which gets annotated by the *DockerWrapper* contains all information to recreate the docker container which annotated the document. By this every document annotated with a *DockerWrapper* is a able to create one or many docker containers from itself which recreate the original environment as good as possible. In fig. 1.2 the core architecture of UIMA is described. One can clearly see that the user defined part of UIMA takes place in the Analysis Engine. Exactly this part can be fully wrapped by the *DockerWrapper* which enables the aforementioned benefits for most user written code in the context of UIMA. In addition to the new capabilities a throughout analysis shall prove that the implementation has an acceptable performance and memory overhead on the examined corpora.

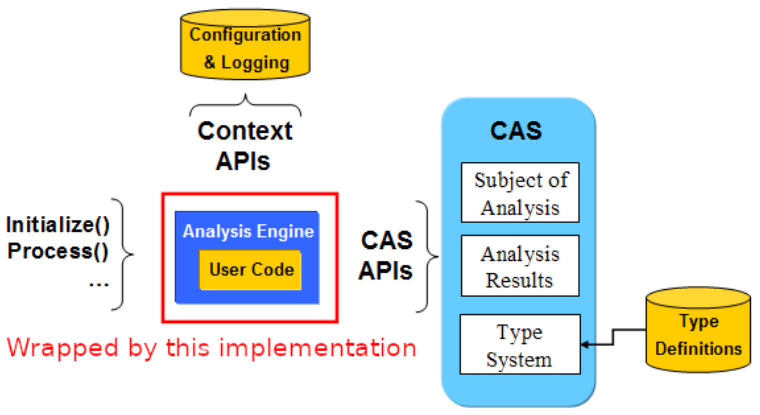


Figure 1.2.: The UIMA Core taken from the UIMA website(UIMA), with a modification to show the wrapped component of the framework

1.2. Contributions

The thesis provides a convenient way to annotate documents with UIMA in a containerized environment improving reproducibility by checking the dependencies on completeness and minimizing operating system differences in a portable and performant way. A novel solution to annotate the used annotators and environment in every document which is processed by this implementation transforms every UIMA document in a self sufficient container builder which can reproduce the exact environment it was annotated with. The annotation of the used dockerfile, analysis engines and additional resources extend the capabilities of the implementation even further. Evaluation of the provided solution shows the feasibility of the proposed approach. The implementation has multiple possibilities to scale on a server cluster. Either a docker swarm network can be used to scale the built container or apache UIMA DUCC[2] can be used to deploy the serialized version of the DockerWrapper to a cluster network. At last an throughout comparision against state of the art UIMA based frameworks reveals that the thesis closes a previously existing technological gap.

1.3. Outline

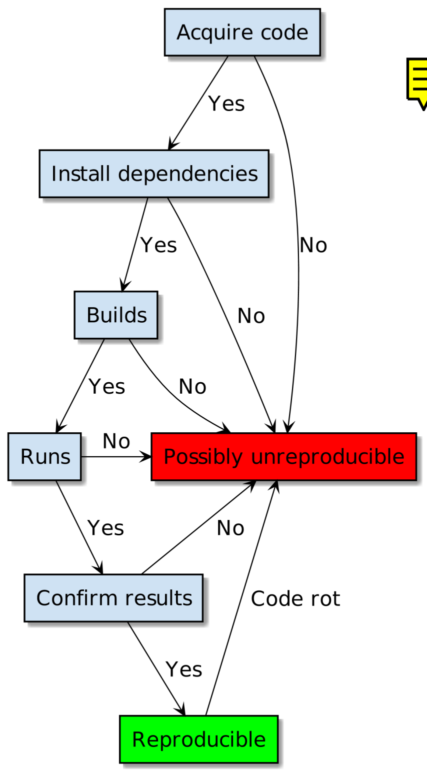
The chapter 2 provides an overview of the current research in the field of reproducibility and a few suggestions made by multiple authors found their way into the recommendations presented in section 1.1. The third chapter reviews some of the current state of the art frameworks for UIMA and their capabilities when it comes to the field of reproducibility. The thesis presents an implementation in chapter 4 after establishing the need to provide another solution to improve upon the shortcomings of the existing frameworks. By reading this chapter the reader should get a good grasp of the inner workings of the implementation including the capabilities and shortcomings. After attaining a general idea of the inner workings, a discussion of the performance and memory implications are performed in chapter 5. The outlook provided by chapter 6 concludes this thesis and envisions improvements in the provided solution by extending with lessons learned from the block chain technology and advanced cluster integration.

[2]https://uima.apache.org/doc-uimaducc-whatitam.html, last accessed at 12/27/2021 22:44 MET

2. Reproducibility research

This chapter reviews the literature on reproducibility in general and considers the lessons learned in this section in the chapter 4 by trying to evade or lower the most common barriers to reproducibility.

Leaning on the definition of the two types of reproducibility the fig. 2.1 shows the life cycle of *reproducible annotations on a different system.*



There are many steps along the way that may fail which highlights the complexity of reproducing previous results in a different environment. Code rot is an integral problem to reproducibility, even if the software ran successfully on a different environment the unavailability of once available resources in addition to operating system evolution may change that.

Figure 2.1.: Reproducible annotation life cycle transferred to a different system.

The following list of barriers to reproducibility is obtained after reviewing the literature on the subject and extracting the barriers most common to the publications.

1. **Code and data availability:** Based on the work of (Collberg et al., 2014) a common barrier to reproducibility is the ability to obtain the code and/or data to even attempt to reproduce the results. The study found that after paper exclusion a mere ≈ 29.1% included code in the form of a web address or in the paper itself. Other authors doing research on reproducibility have excluded all papers without code before even starting with their analysis(Boettiger, 2014). A recent paper concluded from a random sample of 50 papers from EMNLP'18 that the percentage of papers with code remained roughly the same with 30% of the papers including code (Dodge et al., 2020). Even though the numbers have not risen significantly, recent advances in notable conferences like the ML Checklist (Whitaker, 2020) adopted by EMNLP'20 and AAAI'21 additional to the code submission policy

introduced in the reproducibility programme of NeurIPS'19 (Belz et al., 2021) enforce code submission as part of a good publication. The availability of code is therefore integral to the ability to reproduce experimental results.

2. **Installation and build process:** After acquiring the source code the team can start to reproduce the results. As found out by (Collberg et al., 2014) and independently confirmed by (Boettiger, 2014) even when code is supplied the percentage of software that builds and runs without error is below 50% *(n=231)*. This reveals how error prone the build and run process of software itself can be. The installation of the dependencies alone can pose a serious challenge and complex tools like pip[1], conan[2] or maven[3] have been written to deal with dependency resolution. "Dependency Hell" as termed by (Boettiger, 2014) describes adequately the problems stemming from resolving dependency issues which can arise even with the use of the aforementioned tools.

3. **Missing or inadequate documentation:** The documentation of the software itself is crucial for researchers in the process of understanding the implementation made by another team. "Even small wholes in the documentation may pose a significant barrier" for researchers to reproduce the results as pointed out by (Boettiger, 2014). The documentation is even more important in the case of an incomplete code base or a missing dataset in which case the researcher is completely dependent on the documentation to reproduce the missing steps to pre-process build and analyse a dataset. Outdated documentation is also identified as one of the main sources of code rotting (Jun-Yian, 2017) so providing an good documentation may improve upon several steps of the reproducibility life cycle described in fig. 2.1.

4. **Missing adoption as a barrier:** Learning new software libraries and adopting new techniques is a lot of work. No widespread adoption of any technique has taken place despite numerous ways to implement reproducible software by virtual machines, containerization software or a strict documentation coupled with a work-flow paradigm. The ability to reproduce the results therefore does not only depend on the aforementioned points but can be enhanced by a widespread adoption to decrease the work that has to be put into learning a new reproducible software or workflow (Boettiger, 2014).

5. **Code rot:** Code rot describes the incrementally deteriorating performance, maintainability or extensibility of software. One of the most common reasons for code rot is an outdated documentation. When updating the version of a dependency it is easy to forget to update the documentation on which package versions to use. Frequent refactoring and test driven development are able to prevent or slow down code rotting. As can be seen in fig. 2.1 even once reproducible code can become unreproducible when dependency links decay (Jun-Yian, 2017).

2.1. Legal considerations

The legal aspect of distributing software and possibly data needs to be considered as well as pointed out by (de Castilho, 2014). In many cases a program is assembled from multiple libraries which may have a restricting license prohibiting the distribution of any software using the aforementioned library. Even more common in NLP is a license on

[1]https://pypi.org/project/pip/, last accessed at: 12/27/2021 23:04 MET
[2]https://conan.io/, last accessed: 12/27/2021 23:05 MET
[3]https://maven.apache.org/, last accessed: 12/27/2021 23:06 MET

the data which was used to train the model. This is a major problem when publishing experimental results which aim to be reproducible and is one of the main barriers which this thesis had to overcome.

Hereby the software may only be distributed with the consent of the author.

The example license above shows how a license could prevent the code from being published without consulting the author of the aforementioned license before.

2.2. Influence on design

The section before influenced the design of the presented solution here extensively. Reproducible annotations can be implemented in a variety of ways so the following list should provide a justification of the techniques used in the implementation.

Docker

The usage of docker should address the most common problems, the *installation and build process (2.)* and the *code and data availability (1.)*. A docker container can contain all the code it needs to run in addition to all dependencies and all external data (when prohibiting mounted volumes). This enables the researchers to upload the container either to an intern docker swarm[1] network or to docker hub to make the research available to the rest of the world. Additionally (Boettiger, 2014) arguments that the dockerfile itself is a form of *documentation (3.)* by describing all external dependencies that are needed in addition to providing an description of the used operating system and environment. Since the docker container itself annotates which pipeline was used to create this container it is impossible for the container code to become detached from the annotation it makes on the document.

HTTP REST server

By using a HTTP REST interface to communicate with the docker container it is possible to scale the container in a docker swarm or in an AWS server cluster. Additional to the previous benefits its a well known interface and is used by many companies around the world hopefully decreasing the inhibition to *adopt (4.)* the new software. The communication via an network based protocol also enables averting licensing issues. One can use the provided solution and open it up as an webserver enabling other researchers to access the written software without violating code or data distribution laws, *see section 2.1.* The competitor a plain TCP socket which would have been superior in performance and latency has been ruled out due to the previously mentioned simplicity of the HTTP interface and its superior usage in web technology.

HTTP Webview

Every created docker container ships with a web view accurately describing the wrapped pipeline and the wrapped parameters which is a form of automatic *documentation (3.)* which can never get detached from the annotations it creates. In addition to providing the *documentation (3.)* it slows down code rot due to the tight coupling of code and documentation.

[1]https://docs.docker.com/engine/swarm/, last accessed: 12/27/2021 23:35

DockerWrapper design

The design of the DockerWrapper to take an analysis engine description, transform it and return an analysis engine description back is a design choice to lower the barrier of *adoption (4.)* even further. Due to type equality of the input and the output it is simple to just drop in the provided solution and improve the reproducibility of an existing system.

3. Related work

This chapter will review the current state of the art programs using UIMA and comparing the capabilities that they provide with the implementation provided by this thesis.

A large suite of NLP tools is provided by DKPro, originally created by the TU Darmstadt and converted to a community project in 2014. The papers (de Castilho, 2014; Eckart de Castilho and Gurevych, 2015) describes the naming convention of models and resources which allows the program to dynamically load the correct resources at runtime. As described by the aforementioned paper the prefered way to improve the reproducibility of pipelines using DKPro is the creation of a self contained script written in the programming language groovy. Scripts written in groovy contain the dependencies and artifacts needed to run them in the script, by this the script is completely self contained and allows complete fetching of all needed resources on a different system. Since DKPro is a library which can be used it does not include containerization in the standard implementation but users are free to wrap the libraries in a docker container themselves. There is a more recent addition to the DKPro suite addressing improved reproducibility of deep learning experiments (Horsmann and Zesch, 2019). They specifically mention missing containerization as one of the limitations due to the complexity of the dependency set up. This shortcoming highlights the inability of DKPro to provide a unified installation of programming language independent tools even though it provides a sophisticated way to install java dependencies. Since the capabilities of the script are mostly confined to the java programming language, one cannot say that the library is multi programming language friendly *(a)*. DKPro does not provide containerization leading to a poor abstraction of the operating system and the programming environment *(b)*. The scaling is supported with an hadoop cluster[1] *(c)*. Since DKPro supports the self contained groovy script they provide annotator and parameter discovery and composable experiments *(d)*. Changing a dependency later on without updating the documentation on which version created the data is possible. Therefore DKPro does not abide to the tight coupling of data and code so *(e)* is not supported.

Another well known program is cTakes (Savova et al., 2010) a NLP program with a specialisation on clinical texts. The cTakes System relies on UIMA and OpenNLP to create rich semantic annotations. cTakes supports a varitey of pipelines with the ability to export and reimport the pipelines on a different system. Many analysis engines are already provided and can be exported and shared via the self created Piper file format. Installing additional analysis engine components which are not in the installed repository requires manual installation and fixing of the java classpath in order to get the classes into scope. Neither is cTakes multi programming language friendly nor is it operating system and environment agnostic. When extending the default configuration of cTakes with custom Annotators there is no protection against operating system specific behaviour and the installation of python components and inclusion into the framework is not possible. Despite that the program enables easy composability with a GUI that enables the user to compose any number of pipelines. By default the cTakes implementation does not scale very well on a massive corpus. There are two recent papers trying

[1]https://github.com/dkpro/dkpro-bigdata

to improve on the former problems of cTakes. The first (Digan et al., 2021) suggests a workflow management system WMS to improve repdocucibility in 7 clinical NLP suites one of which is cTakes. Containerization is not standardised in cTakes but there are several repositories allowing a containerized version of cTakes[1]. A recent paper (Digan et al., 2021) examined how to improve reproducibility of several clinical NLP frameworks one of which was cTakes. They specifically mention missing features like relative path resolution within tools, which can lead to problems when transferring tools to another system. In addition to that there is no version control for tools or resources. cTakes in the standard version does not use containerization but since a containerized version is available online the multi programming language friendliness *(a)* and the operating system and environment agnostic capabilities *(b)* are at least partially fulfilled. It is scalable by default *(c)* and supports the exporting and composing of different pipelines which are stored on the local system, enabling composable reproducible annotations *(d)*. Despite that there is no tight coupling of data and code which may lead to reproducibility issues therefore *(e)* is not supported. There is one thing to note though, installing cross programming language tools may require rebuilding of the whole container and the configurations between different container setups may not be interchangeable.

TextImager is another relatively new tool providing extensive NLP annotation tools(Hemati et al., 2016). The NLP suite is containerized based on the github repository of the TextImager server[2]. Therefore it provides a sandboxed runtime environment which is able to abstract most operating system differences away. Extending the TextImager with custom classes requires the user to copy the compiled JAR Files into the TextImager container or adapting the dockerfile to download the user defined JAR files into the container. After examining the client of the TextImager[3] it becomes clear that the current annotator configuration is not obtainable for all tools. Many annotators are exposed on a specific IP address which enable the TextImager to scale well on multiple servers but hides away the annotator configuration which may be subject to change ultimately defying the possibility to reproduce the results even when using the same configuration. Due to the containerization the TextImager fully abides to the multi programming language friendly setup *(a)* and also to the operating system and environment agnostic capability *(b)*. It is scalable by leveraging the apache DUCC setup *(c)* but misses out on the points annotator and parameter discovery *(d)* as well as composability of the reproducible annotations *(e)* which depends on being able to create one reproducible experiment.

The last tool to compare against is the UIMA PEARs (Processing Engine ARchive) standard. It is described as the "standard packaging format for UIMA components like analysis engines (annotators)" apache UIMA. UIMA PEARs provides all the tools to deploy a UIMA component to another computer including additional resources and scripts. Since the packaged components are fully compatible with the UIMA DUCC system it is possible to easily scale the solution in a server cluster *(c)* and composability is inherent to any uima analysis engine component due to the ability to compose any number of analysis engines to an aggregate analysis engine. The setup is neither environment nor operating system agnostic, but it has a decent support for cross platform usage due to the interoperability of java bytecode but this does not abstract away a lot of the system settings *(b)*. It is also not multi programming language friendly since one can package components which are not inherent to the java programming language but the necessary

[1]https://github.com/choyiny/ctakes-server
[2]https://github.com/texttechnologylab/textimager-server
[3]https://github.com/texttechnologylab/textimager-client

tools to use these files on another system must be installed by the user. This is especially important when creating platform specific code for example in the language C++, Rust or Assembler. Therefore the point *(a)* is not well supported by the uima PEARs package. As for all other compared frameworks point *(e)* is not supported, this is a unique trait of the presented solution here which is not present in any other examined system. To our knowledge there is no system leveraging the UIMA system to create self reproducible annotations from the UIMA system.

Tool	**(a)**	**(b)**	**(c)**	**(d)**	**(e)**
DKPro	−	−	+	+	−
cTakes	•	•	+	+	−
TextImager	+	+	+	−	−
uima PEARs	−	−	+	+	−
DockerWrapper	+	+	+	+	+

4. Implementation

We introduce *DockerWrapper* to improve upon the aforementioned shortcomings of existing software solutions. The internal handling of the wrapped UIMA Analysis Engine is described in fig. 4.1. An analysis engine is the basic building block of most UIMA based pipelines and analysis tools. It is used by cTakes (Savova et al., 2010), TextImager (Hemati et al., 2016), TextAnnotator (Abrami, 2021) and most other UIMA based tools. So in order to widen the scope of this implementation it works by wrapping one of the smallest building blocks of an UIMA pipeline, the uima analysis engine description. By containerizing the given analysis engine in an automated way it is possible to use docker based tools to create an experiment which can be scaled easily. As can be seen in fig. 4.1 the *DockerWrapper* can scale by using either DockerSwarm or apache UIMA DUCC. One important thing to note is that the DockerWrapperContainerConfiguration is not saved alongside the dockerfile information inside of each CAS document. This is due to differing hardware configuration which should not impose restrictions on which annotations can be reproduced. In theory any change made to the DockerWrapperContainerConfiguration should not change the behaviour of the wrapped analysis engine inside of the container. Using or not using a GPU may impose such differences for the analysis engine so it should be used with care. The usage of the implementation with these tools have been tested and some utility functions are supported to improve the integration into these tools. The communication with the docker container is through an HTTP REST Interface which is exposed by every wrapped container. This enables the usage of load balancers and the distribution and orchestration of containers without UIMA specific tools but by using means of scalable web technology like Kubernetes or AWS.

A *DockerWrapper* itself is a UIMA Annotator. By this it can be serialized to and from an XML descriptor, combined into an primitive or aggregate analysis engine and used by UIMA-AS and UIMA DUCC for scaling. The approach to implement the *Docker-Wrapper* as an analysis engine has numerous benefits and improves the interoperability with other UIMA related tools. Based on the research of reproducibility from chapter 2 adoption is a common barrier to reproducibility. In order to increase adoption a simple HTTP Server with a REST interface is used to expose the container functionality to the outside. Both technologies are well known and largely adapted in many programs and by this are hopefully lowering the adoption barrier.

The provided fig. 4.2 shows the interaction of the software implementation with the basic UIMA components and may highlight how tighly the implementation is coupled with the basic building blocks of UIMA. One thing to note in the figure is that by the design provided for this architecture it is possible for a *DockerWrapper* to contain another *DockerWrapper* which can then contain even more *DockerWrapper* and all of these could be serialized to XML and back again. This highlights the flexibility of the provided solution but this flexibility brings also a few new problems which are addressed later on in the thesis. One of the problems is the *deep nesting problem*, section 4.2, arising from the inability to nest docker containers inside of each other. The implementation provides the possibility to flatten deep container structures to prevent this from happening.

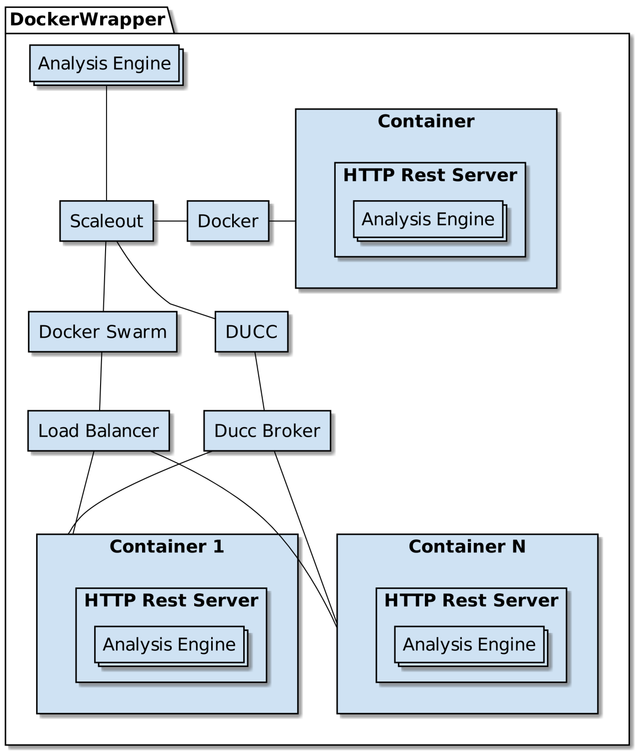


Figure 4.1.: An schematic overview of how the DockerWrapper wraps a given UIMA Analysis Engine.

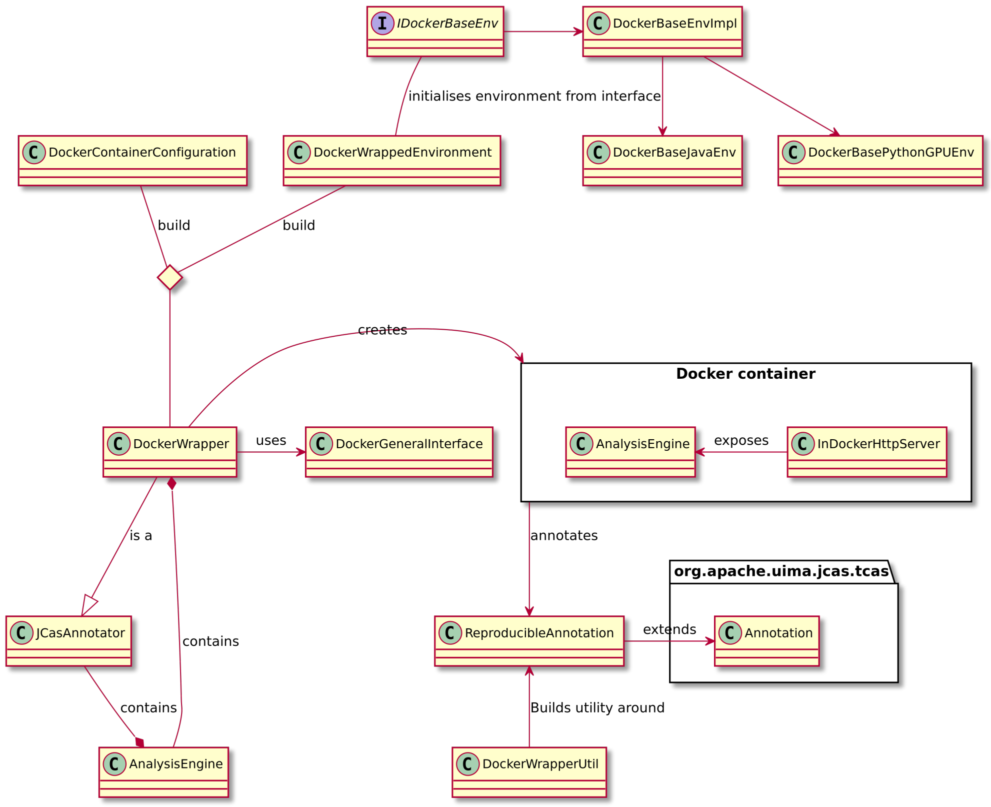


Figure 4.2.: The class relations of the *DockerWrapper* and the related classes to the UIMA framework. This figure shows only the most relevant classes to improve the readability.

4.1. Design

A well known implementation of the UIMA System is written in the Java Programming language [1] which is used for the implementation of *DockerWrapper* as well. Only slowly new programming languages gain access to UIMA which is and has been a domain almost exclusive to the Java Programming language. The general paradigm of the design is to create frictionless code which can easily integrate in larger software.

The flexibility of the solution stems from the ability to wrap an arbitary amount of analysis engine descriptions into a single AnalysisEngineDescription which contains the DockerWrapper and the other analysis engine descriptions. By this one can serialize, deserialize and modify the DockerWrapper as any other Annotator in the UIMA framework. This also enables the feature of deep nesting, which describes a possible setup in which a DockerWrapper contains multiple other DockerWrapper which then contain multiple analysis engine descriptions. As docker does not enable running docker containers inside of docker containers this presents a serious challenge to the implementation. This is overcome by an workaround which is described in detail in the section 4.2.

4.1.1. Docker

Docker is used by a variety of applications and gained attention specifically for its low overhead containerization. It is used extensively in the industry to provide continous deployment and microservices (Anderson, 2015). Continous deployment faces similar issues as reproducible annotations since the underlying problem is developing a program in one environment and trying to set it up almost frictionless in another production environment. Using docker to improve reproducibility is well known from the research. Already in 2014 (Boettiger, 2014) used docker container to improve the reproducibility of R programs by setting up the whole Editor inside of a docker container. More recently (Boettiger and Eddelbuettel, 2017) improved upon the previous work by providing specialised environments for the development with the R programming language. Even for web development (Cito and Gall, 2016) is docker a viable tool to improve reproducibility. Addional to the previously mentioned research the first and only collaborative study on reproducibility in the context of NLP (Branco et al., 2020) required the participants to submit a docker container with their results. Despite the overall positive effects mentioned by most papers there are also critical views on docker as a technology to facilitate reproducibility. (Rougier et al., 2017) criticised that docker may provide an portable executable but cannot fix missing or inadequate documentation. In addition to that one of the criticised points concerns the stability of the docker container file format since the technology itself is more geared toward deployment with “no visible ambition towards archieving computational environments”. This is a valid concern but docker containers are based on the OCI[2] (Open container initiative) which enables interoperability with different container providers. Since some of them for example podman[3] are hosted on github there are ways to maintain dated containers even if the container file format may change in the future. This means that docker container can be used by podman and vice versa. The whole previously mentioned research shows that docker is a fitting technology for improving reproducibility and is used in the thesis for this reason.

[1] https://uima.apache.org/
[2] https://opencontainers.org/
[3] https://podman.io/

4.2. Deep nesting

Deep nesting describes the problem of having a DockerWrapper inside of another DockerWrapper implementation. This is a serious problem as docker disallows multiple containers nested inside of each other. To overcome this problem the implementation is able to recursively map the host docker socket inside of the child containers. Any nested DockerWrapper will spawn a sibling container instead of a child container leading to a flat hierarchy which is allowed by docker. The implementation is able to detect if the engine is executed in an docker container or not. This is important for the IP address mapping which must be dynamically resolved based on whether the implementation runs in a container or not. The IP address of the host is resolved by extracting the information from the docker gateway bridge network.

Security implications of deep nesting

There are several security implications involved in mapping the docker socket inside every child container. A direct access to the docker socket equals root access on most systems. The docker daemon runs as root on most systems and a malicious attacker may exploit this by creating a new container which maps resources which can only be accessed by the root user. Recently docker introduced a rootless mode which decreases the attack surface. Using the recursive mapping in rootless mode has no security implications beyond running a possibly unknown program on the computer. As the security implications of this solution are severe it is disabled by default and must be enabled manually.

4.3. Graphical user interface

A graphical user interface has been implemented to view the container configuration, the wrapped pipeline, the resources associated with it and the used type system. This should help with the integration and documentation of remote containers and local containers as well. It is not possible to change the analysis engine configuration from this interface. This is a design decision in the spirit of reproducible annotations. Changing the setup of a once configured container without building a new container leads to a mutable state inside of the container which may lead to research not being reproducible since the underlying container has changed.

The following list provides an overview of all screens and the documentation that they provide for the user. This graphical user interface is accessible for every container built with this implementation under *http://containerip:containerport/documentation.*

- **Menu** - The menu is the basic navigation element of the provided documentation. It has links to all other pages, an example can be seen here fig. 4.3.
- **Information** - The main documentation page should provide the user with an overview of available api endpoints. It looks similar to the spark documentation but has not the same capabilities. It is purely a visualisation of the provided API endpoints. fig. A.1
- **Analysis engine** - The analysis engine screen provides an overview of what engines are wrapped and what configuration these engines have. On the screen we can see the SoFa mappings, parameter settings and all used annotators. In addition

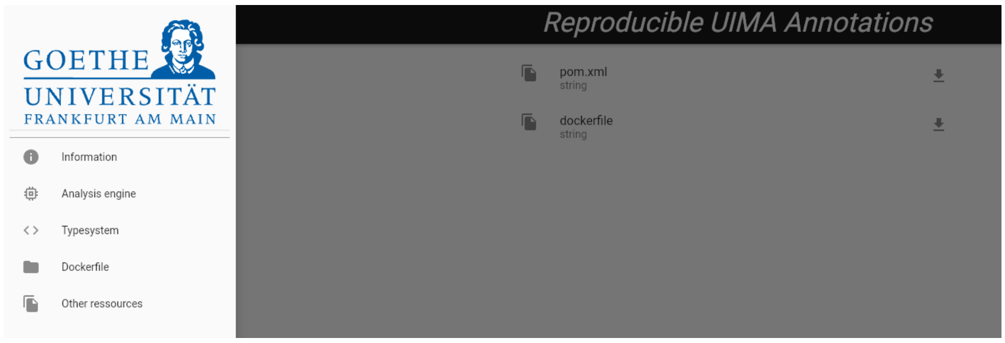


Figure 4.3.: The main menu of the graphical user interface, showing all possible screens that can be accessed by every container.

to that it provides recursive resolution of aggregate analysis engines to provide this informations even for nested analysis engines. An example can be found here fig. A.3.

- **Typesystem** - The typesystem screen shows a view of the typesystem used for this analysis engine. The tree is created by resolving and recursively mapping all supertypes till no more resolution is needed. An exemplary view can be found here fig. A.5.
- **Resources** - This screen shows all resources used to create the container. A user is able to download every resource seperately to inspect it fig. A.4.

The implementation of the graphical user interface has been written with the tool flutter[4]. This tool enables an simple way to migrate the given code to either iOS, Android or a Website. An example view of the mobile version look can be found in the appendix fig. A.6. The code can be found in the github repository of the thesis[5].

4.4. Guarantees

A few guarantees can be made by the implementation through careful engineering of the software implementation.

Integrity of the reproducible CAS document
The implementation calculates the CRC32 checksum after every DockerWrapper and checks the last CRC32 checksum before starting to process the CAS document. This ensures with almost near certainty that no other annotations have been made or removed which are not covered by such an implementation.

Temporal integrity of the document
Every *DockerWrapper* annotates the CAS document with the used pipeline and the time stamp as well. The implementation provides convenience methods to restore the used pipelines either from a single view in chronological order or from the aggregation of all views. The software guarantees that the pipelines will be restored and ordered just the way they were executed on the underlying CAS document. The time stamp is saved as milliseconds since epoch which is synchronised and a common format for every major

[4]https://flutter.dev/, last accessed: 12/28/2021 00:15 MET
[5]https://github.com/texttechnologylab/ReproducibleUIMAAnnotations

operating system. There is no protection against malicious tampering with the CAS documents, this is out of the scope of the thesis and it is expected that researchers do not maliciously tamper with their own experiments. This implementation can fail if the time is wrong on the executing operating system which happens rarely in a time where time servers are used to keep the clock of almost every computer synchronised. Nonetheless it shall give the reader an overview of the capabilities and pitfalls of the implementation.

4.5. Performance and space optimizations

A lot of time has been invested to make the performance and the additional space consumption feasible for an per document basis. An throughout evaluation can be found in the Evaluation chapter 5 which will discuss the trade off between performance and space consumption. In order to recreate the analysis engine settings as good as possible the whole XML representation of the analysis engine is stored in the CAS document. To give an rough overview a single analysis engine with type system serialized to the XML representation can take up to 1MB of memory. When considering the case of annotating tweets which have a low average text length the space consumption of the tweet may increase by more than a factor 1000 for one annotated pipeline. To decrease the space pressure the thesis uses well studied compression algorithms. In addition to this approach a bit of interaction of the user is needed to improve the space consumption even more. Most of the space consumption is attributed to large type systems which scale very fast due to the type system auto detection of UIMAfit which includes every type in the class path to build the main type system. When using only a small subset of types from the global type system it is wise for the performance and the space consumption of the implementation to limit the type system of the analysis engine to the types used inside of the analysis engine. A lot of effort is put into optimizing partial type system handling.

Performance and space benefits of using a partial type system:

- **Serialization** - Since in UIMA the whole type system used by the analysis engine is serialized in addition to the analysis engine, limiting the types to the types actually used reduces the serialisation size dramatically.
- **Network** - The implementation only serializes feature structures which are used in the remote type system from the analysis engine. When using only a small subset of types the resulting communication puts a lot less strain on network and CPU resources.

Further performance improvements:

- **Delta serialization** - For Java implementations only the delta of the CAS document, consisting of newly created annotations is returned to the callee.

4.5.1. Load balancer

Load balancers are native to kubernetes, docker swarm and nginx. The load balancer tries to distribute the incoming traffic to the underlying services, in a way which does not overload one of the services (Mohan et al., 2020). Many load balancers for a cross server setup work by resolving the DNS request to the least used server IP address. The roundtrip over the load balancer may consume a bit of time which throttles performance in the case of small document corpora or simple pipelines. To overcome this problem the docker swarm integration manages a pool of connections which use HTTP/1.1 keep alive to persist a once established connection, avoiding the roundtrip over the load balancer apart from the first connection establishment. This improves the latency and runtime of the tested corpora. Since the number of containers replicated across the whole swarm is less or equal to the concurrent requests made against that network, there is no need for load balancers in this setup.

4.6. Minimal example

This section provides a minimal working example to enable the reader to better capture the simplicity and flexibility of the provided solution.

```
// Container configuration, autoremove, gpu support, deep nesting feature, scaleout...
DockerWrapperContainerConfiguration cfg = DockerWrapperContainerConfiguration.default_config()

//The engine to wrap, notice this is an AnalysisEngineDescription
AnalysisEngineDescription engine = AnalysisEngineFactory.createEngineDescription(ExampleAnnotator.class,ExampleAnnotator.PARAM_PIPELINE_CONFIGURATION, "");

//Wrap the given engine, can be primitive or aggregate AnalysisEngineDescription
DockerWrappedEnvironment env = DockerWrappedEnvironment.from(engine);

//Creates an AnalysisEngineDescription again, serializable and usable the same way as the engine before
AnalysisEngineDescription wrappedEngine = env.build(cfg);
```

Listing 4.1: The sample code to create use the implementation

Listing 4.1 shows a minimal example of how to use the provided implementation. More examples can be found in the tests of the provided solution. This example should provide the reader with an overview of how easy it is to integrate the docker wrapper into an application.

```
//jc should already be annotated with some reproducible annotations
JCas jc;

DockerWrapperContainerConfiguration cfg = DockerWrapperContainerConfiguration.default();
//Recreates all analysis engines with the given container configuration
AnalysisEngineDescription reproduced = DockerWrapperUtil.fromJCas(jc,cfg);
```

Listing 4.2: The sample code to recreate reproducible annotations given a JCas Object

Given any CAS document which was annotated by one or many DockerWrapper the reproduction of the annotations is a simple two line process. Notice though that the configuration of the DockerWrapperContainerConfiguration is applied to every reproducible annotation which is found in the document. The solution provides also more sophisticated methods to change single annotators or the DockerWrapperContainerConfiguration of a single reproducible annotation.

4.7. Security

The DockerWrapper implementation has the same security benefits as any other docker container. The code runs in a containerized environment and when not making use of the deep nesting feature (section 4.2) the code runs in a sandboxed environment. Every reproducible annotation annotates the used "dockerfile" to create the container where the annotators are run. A security aware application only needs to check the "dockerfiles" made by the reproducible annotations to access the security implications of running a foreign annotator. Malicious code will not be able to access host resources as long as they are not exposed by the docker file.

Log4j: The implementation uses the logging library SLF4J instead of Log4j, therefore there should not be an attack surface connected to the recent findings of the Log4j zero day exploit (Zhaojun, 2021).

4.8. Failed attempts

This section shall provide a small overview of the other attempts that were made in order to accomplish reproducible annotations. The publication bias first described by (Phillips, 2004) describes the underrepresentation of "uninteresting" or failed attempts in research. This leads to a shift in published papers reporting mostly successful experiments. Reporting failure too on the other hand may result in less researchers trying to use a technique which has failed beforehand. This is why there is a small section in this thesis describing the attempts that did not work out for various reasons.

Runtime detection of dependencies: Using the class loader it is possible in java to find all loaded classes and by introspection it is possible to get the package version and the package name. Despite that it is not always possible to determine the package version and it is not possible to translate package version and name to an artifact that can be downloaded. It may be possible when using exclusively maven, still this is no feasible approach when using just java and certainly not when using a multi programming language environment. The solution is using a dockerfile which automatically tracks all important dependencies. The self validation happens in the build step of the docker container which will abort the build process when a missing dependency is detected.
Type system pruning of analysis engines: One can cut out the analysis engine type system, this drastically reduces analysis engine serialization size. The type system can be collected inside of the container with the UIMAfit type autodetection. There are some large disadvantages to this approach. As was discussed before the implementation uses the analysis engine type system to determine which types to serialize and send to the docker container. When there is no type system in the analysis engine one has to always send the whole CAS document. This reduces the aforementioned benefits greatly. Even when employing a technique to discover the type system from the container, the dangers of using a pruned analysis engine which possibly cannot be handled by UIMA in every case, outweighs the benefits.

4.9. Dependencies

As in most programming projects a multitude of open source software is used to built upon. This section shall provide the credit which is due for these libraries.

- **Latex** - The thesis is written in this excellent language (Mathematik, 1989) oldest published study on the subject of latex from the official website.

- **PlantUML** - Was used to create most of the diagrams in this thesis[6].
- **Java dependencies**
  - **Java Docker library** - Used to communicate with the docker daemon on the system [7].
  - **UIMAJ, UIMAfit, commons compress, OpenNLP** - All provided by the apache software foundation. [8].
  - **SQLite** - Originally provided by (Bhosale et al., 2015) and wrapped by [9].

[6]https://plantuml.com/
[7]https://github.com/docker-java/docker-java
[8]https://www.apache.org/
[9]https://github.com/xerial/sqlite-jdbc

5. Evaluation

The evaluation consists of a comparision against state of the art research in the field of reproducibility and of the evaluation and costs associated with the usage of the implementation.

5.1. Comparision against baseline

The baseline is defined as the unwrapped analysis engine which does not make use of the software solution implemented here. In the following section the baseline is compared against the analysis engine wrapped by the presented software solution. A throughout analysis in terms of performance and storage consumption will be made. A set of different analysis engines is used as a baseline to emulate different use cases and different natural language pipeline complexities. For this purpose 4 different baseline are chosen they are composed of either 1,2,3 or 4 different annotators. The annotators are used always in the same order to honor inter-annotator dependencies.

Number of annotators	Annotators
1	BreakIteratorSegmenter
2	BreakIteratorSegmenter, CoreNlpPosTagger
3	BreakIteratorSegmenter, CoreNlpPosTagger, CamelCaseTokenSegmenter
4	BreakIteratorSegmenter, CoreNlpPosTagger, CamelCaseTokenSegmenter, CoreNlpNamedEntityRecognition

Table 5.1.: The mapping of the number of annotators to the used pipeline for the evaluation. The annotators are provided by the apache OpenNLP library (Apache Software Foundation, 2014)

5.1.1. German political speeches corpus

Due to computational limitations the implementation was tested on the german political speeches corpus from 2019. The presented corpus is a collection of many political speeches held in the parlament or due to offical matters. It consists of over 6500 different speeches and more than 10 million words. The corpus is monolingual and only contains texts in the language german.

Trait	German political speeches
number of documents	6685
mean [bytes]	11975.684518
median [bytes]	9596.0
standard deviation [bytes]	8056.441047
smallest document [bytes]	26
largest document [bytes]	70230

Table 5.2.: This table provides a short overview of the german political speeches corpus

One document which is part of the dataset only has a length of 26 bytes. This is due to the extraction program, appendix A.3, cutting out all the metadata and only keeping the text. After cutting out the metadata all thats left for this document is the short text: “>> Die Rede im Wortlaut.”.

The corpus has a relatively large average document size compared to the wikipedia corpus. In contrast to the wikipedia corpus the median does not deviate that much from the average document size indicating a smaller number of outliers or a smaller deviation from the average document size for this corpus.

Performance comparision

This is the performance comparision between the DockerWrapper and the baseline pipelines on the german political speeches dataset.

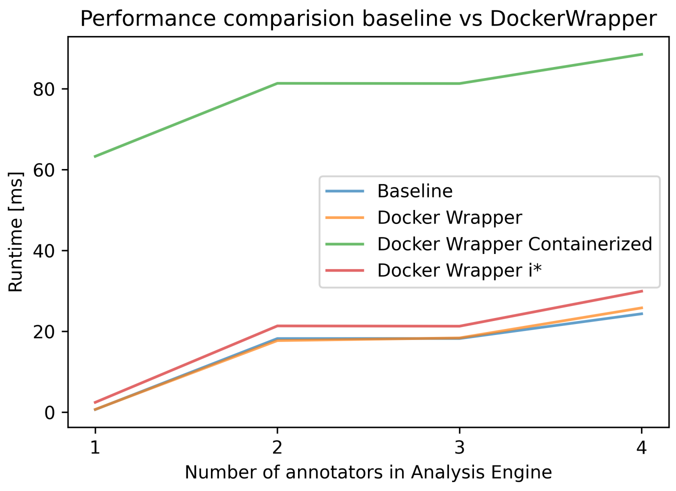


Figure 5.1.: The average runtime in milliseconds of the different implementations

One can clearly see from the diagramm that the computational effort of the Docker-Wrapper implementation is for the containerless version very close to the baseline for all pipelines used. When additionaly opting in the integrity check of the DockerWrapper denoted by the i* at the end, the performance drops considerably. This is due to the need to serialize the returned CAS document to compute the checksum on the CAS. The containerized version of the DockerWrapper is much slower than the baseline. Since the docker container uses a REST interface and passes the whole CAS document through this interface to the container 2 complete serializations and 2 complete deserializations are needed. This explains the slower performance in the containerized environment. The performance slowdown in a clusterized environment may be insignificant, as the sending of CAS documents between the clusters is necessary anyways so there should not be any additional overhead involved in using the DockerWrapper implementation.

Storage consumption

This is the storage consumption comparision between the DockerWrapper implementation and the baseline analysis engine.

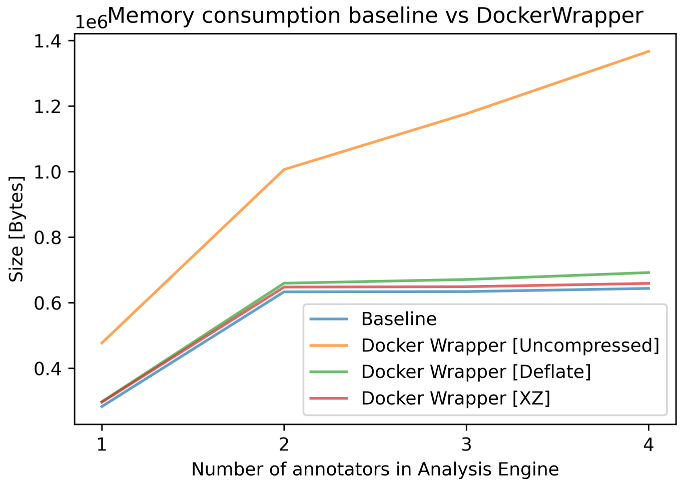


Figure 5.2.: The average runtime in milliseconds of the different implementations

From the fig. 5.2 one can see the uncompressed implementation of the DockerWrapper leads to a great increase in the average document size. Since the implementation stores the whole used analysis engine as well as the dockerfile and the pom file of the user, the document size increases linearily with the amount and complexity of the annotators used. To get the storage consumption down to a reasonable size an compression algorithm is needed. The implementation supports multiple compression algorithms by means of the apache common compression package. As one can see the usage of the XZ compression algorithm amoreliates the great increase in average document size.

5.1.2. English wikipedia corpus sample

Another corpus which was used is a sample from the english wikipedia. The corpus is very popular in the field of NLP due to the license of Wikipedia and the general quality of the entries. Since the full english wikipedia corpus has around 5 million texts with over a billion words the corpus used here is only a sample of 20000 uniformly picked documents from the original corpus.

Trait	Sampled corpus	Full corpus
number of documents	20000	4641026
mean [bytes]	1829.878650	1795.605
median [bytes]	695.5	681.0
standard deviation [bytes]	3733.277978	3686.524
smallest document [bytes]	8	1
largest document [bytes]	78664	253700

Table 5.3.: This table provides a small overview of the picked sample.

Note: The english wikipedia corpus contained more than 300000 documents containing only a document title, these documents have been removed before taking the sample and calculating the mean and standard deviation. The code can be found in the appendix A.2.

$$\frac{(1829.88 - 1795.61) \cdot 100}{1795.61} \approx 1.91 \tag{5.1}$$

$$\frac{(3733.28 - 3686.52) \cdot 100}{3686.52} \approx 1.26 \tag{5.2}$$

$$\frac{(695.5 - 681.0) \cdot 100}{681.0} \approx 2.13 \tag{5.3}$$

Notice that both the mean document size, the mean document size standard deviation of the sample and the median is within 2.2% of the original corpus. Another important thing to note is that even though the mean average document size if $\approx$ 1829 Bytes, the median is only 695.5 Bytes. This shows that the outliers in the wikipedia corpus have a large impact on the mean average document size. The standard deviation is also very large due to the outliers. When comparing the median of both provided corpora we can conclude that the wikipedia corpus sample can be categorized as corpus with a lot of small documents and some big documents, and the german political speeches corpus can be categorized as a corpora with many medium sized documents with outliers in both directions favoring the larger documents.

Sample creation

A small program has been implemented in the Rust programming language to extract a random sample of wikipedia texts from the original corpus. The corpus was downloaded from here[1] in the JSON format. Rust was chosen due to its high performance especially for JSON serialization and deserialization. The random sampling of the wikipedia pages is further cleaned by purging all empty pages from the sample. All sampled pages will be saved in the text format together with the collected statistics such as document length in the output directory. The code can be found in the appendix A.2. An example of how to create a sample from the wikipedia corpus is provided in the appendix A.1.

Performance comparision

This section provides a short comparison of the DockerWrapper implementation against the baseline analysis engine. The annotations made by both implementations are the

[1]https://www.dropbox.com/s/wwnfnu441w1ec9p/wiki-articles.json.bz2?dl=0

same except that the DockerWrapper adds another annotation to reproduce the environment and the libraries used to instantiate and use the aforementioned baseline analysis engine.

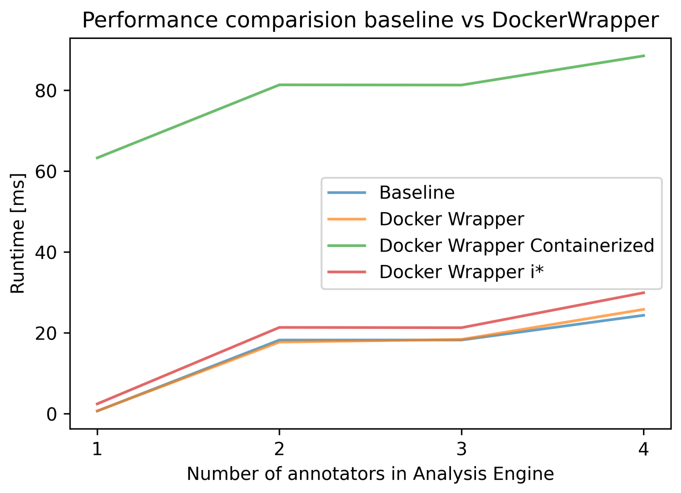


Figure 5.3.: The average runtime in milliseconds of the different implementations on the 20.000 sample documents from the english wikipedia corpus

The performance measures are taken after every iteration of the analysis pipeline. There is no separate evaluation of the different compression methods in terms of performance since the compression takes place at initialization time and the computational effort for the processing of a CAS document is not effected by this. As one can see the performance of the baseline and the basic DockerWrapper implementation are almost on foot. This shows the computational effort to annotate the CAS document with the used analysis engine is negligible. The DockerWrapper i* confirms the integrity of the CAS document on every iteration. For this to work the whole CAS document has to get serialized and annotated one more time, this explains the large difference in performance between the two DockerWrappers. The slowest implementation is the one in the container, since the container works with a REST interface the whole CAS document needs to get serialized and de serialized twice. For executions on the same machine this is a downside of the implementation. On a cluster the serialization and deserialization of the CAS documents is necessary anyways to distribute the CAS documents to the available servers. The performance difference may not be as large in such a setting.

Storage consumption

This section provides a short comparison of the DockerWrapper implementation against the baseline analysis engine. The storage is calculated by converting a resulting CAS document to the XMI representation and taking the length of the resulting XML string.

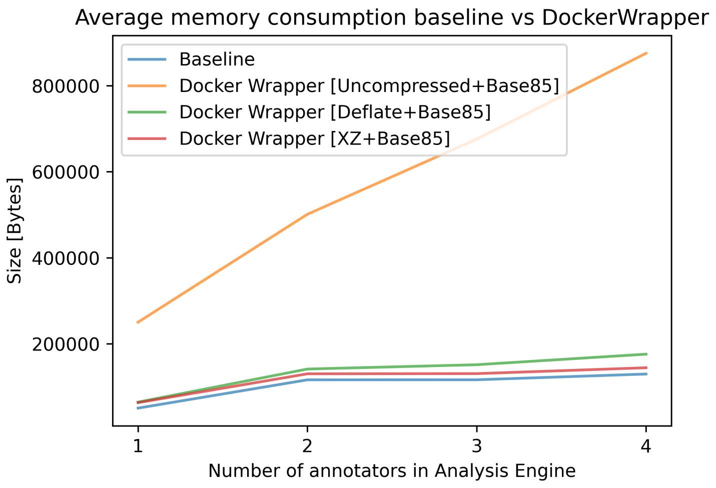


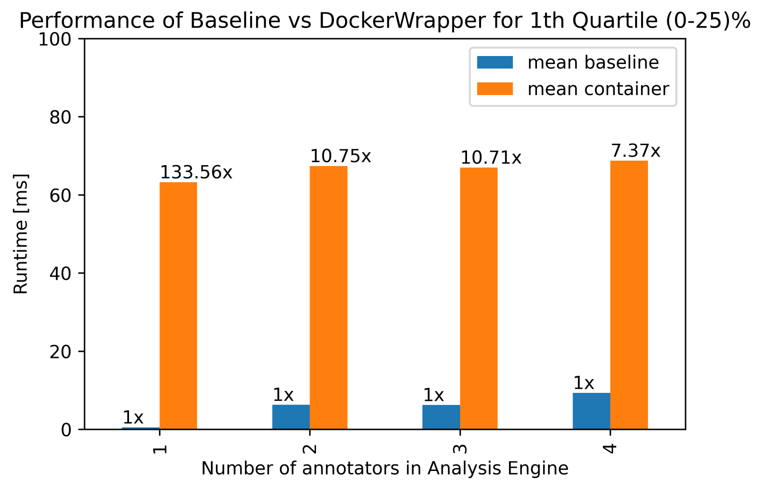
Figure 5.4.: The average document size in bytes for the different implementations on the 20.000 sample documents from the english wikipedia corpus

The evaluation clearly shows that the complexity of the analysis engine affects the mean document size a lot. As the pom file, the XML representation of the analysis engine as well as the dockerfile is saved in every CAS document average size of the CAS document is increasing linearly with the size of the aforementioned files. Every DockerWrapper encodes the configuration in the Base85 format, this averts problems with characters which are not escaped but should have been escaped and also with differences in the default character encoding of the reading implementation clearly shows that the complexity of the analysis engine affects the mean document size a lot. As the pom file, the XML representation of the analysis engine as well as the dockerfile is saved in every CAS document average size of the CAS document is increasing linearly with the size of the aforementioned files. Every DockerWrapper encodes the configuration in the Base85 format, this averts problems with characters which are not escaped but should have been escaped and also with differences in the default character encoding of the reading implementation. To improve the average document size different compression algorithms are supported. The decision which compression algorithm is used is up to the user, this evaluation compares two possible choices with respect to the resulting document size. As one can see the compression greatly reduces the document size to an acceptable level. The usage of the DockerWrapper increases the average document size in every case less than 10%. The compression of the configuration takes place at the initialization time and is reused for every document processed, therefore there are almost no performance implications for the processing time of documents.

5.1.3. Discussion

The DockerWrapper implementation adds an annotation to every document. Only this operation has a neglible effect on performance as can be seen in the figures before. Therefore the main overhead of the implementation is the serialization and deserialization overhead which is needed to transfer the CAS document to the container and back again. As seen in the diagramm this overhead is large and must be considered when using this implementation. Noteworthy is that for clusterized setups which distribute the CAS documents to a network of servers this step of serialization and deserialization is needed anyways. Therefore the overhead could be comparable to the expected overhead of a apache UIMA DUCC server cluster. This still needs to be evaluated but it is obvious that the overhead which decreases the performance of the containerized version is also a great strength of the implementation. Any analysis engine can be transformed into a scalable docker container which can be distributed to a docker swarm network or scaled by an AWS setup. This enables new possibilities for the wrapped analysis engines far beyond the reproducible annotations that they provide. As one can see in the tested docker swarm setup, the provided solution outperforms the basic analysis engine when adding a sufficient number of replicas to it. It is important though that the scalable docker containers are implemented to show the possibilities that a network capable analysis engine can provide in combination with container scaling techniques. There is no ambition to provide an alternative to apache UIMA DUCC and the like, it should just point out that there are unique benefits associated with the increased overhead of an network capable docker container.

A visualization of the overhead can be seen in fig. 5.5, which clearly shows that there is a constant overhead involved in the transfer to the container. This overhead is problematic especially for small documents and for fast pipelines since the contribution of the overhead to the total runtime is larger for those cases. The comparison of the 25% smallest documents with the 25% largest documents shows really well how the constant overhead is diminishing compared to the runtime of the analysis engine pipeline. For every tested pipeline in the largest 25% of documents the slowdown of the implementation is significantly smaller than the slowdown imposed by the implementation for the smallest 25% of documents. Based on the assumption that a single BreakIteratorSegmenter annotator in the pipeline is not the most realistic use example of this implementation one can assume that with increasing complexity of the pipelines the implications on the performance of the analysis engine pipeline decreases more and more. The same conclusion can be drawn when looking at the same diagram from the german political speeches corpus A.7. For this corpus the implementation also performs consistently better for large documents and for more complex pipelines.



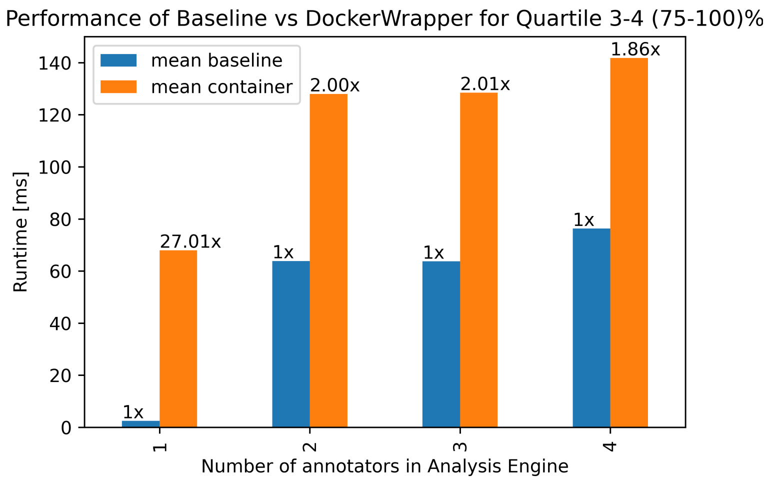


Figure 5.5.: The performance of the containerized DockerWrapper vs Baseline on the english wikipedia corpus based on size quartiles.

In contrast to the performance comparison there is no significant overhead for the average CAS size. This is mainly due to the use of compression algorithms which can reduce the size of the added annotation drastically. It should be noted that the overhead on the CAS size always depends on the corpus which is annotated. As can be seen in the two tested corpora, the average document size is always greater than 1000 bytes. This leads to a large average CAS document size after using the aforementioned annotators on the documents. Since the reproducible annotation size created by the DockerWrapper only depends on the pom,dockerfile,analysis engine, typesystem and the user annotated resources it is constant no matter how large the processed document is. This leads to an large overhead for small documents but a negligible overhead for larger documents. Therefore the added overhead is highly dependent on the chosen setup and corpus. At least for the english wikipedia sample and for the german political speeches corpus the overhead is small.

5.2. Limitations

The provided implementation can be seen as a proof of concept. Many options which are supported by docker containers cannot be accessed from the implementation at this moment. For example there is no ability to map user provided volumes which may be useful in the context of sharing large machine learning models with several different containers. There is also no simple way to use local code in the implementation. This seems like a large limitation but one has to keep in mind that using local volumes and local code is certainly not reproducible on another machine which does not have the exact same file system or volume layout. Therefore no work was put into providing a good solution here as it is hard to make such annotations reproducible. This limitation may be lifted in future releases with specific conditions which enable reproducibility when accessing local resources. Another limitation is the missing usage of SSL connections, this is important to provide cryptographically secure connections to the docker swarm network.

6. Conclusion

The implementation provides a convenient method to increase reproducibility of existing pipelines with minimal effort. Despite the simplicity of the solution it is general enough, through the use of interfaces to adapt the implementation for more complex applications. Still many features are missing for the current implementation and some ideas for future research is examined at the end of this thesis.

6.1. Multiple wrapper fusion

At the moment the annotation of documents by multiple DockerWrapper pipelines leads to multiple reproducible environment annotations on the object. Leading to a linear growth of the document size and a fragmentation of the used annotators in several distinct annotations. When reproducing the results later on the amount of containers needed to reproduce the pipeline is the same as the amount of DockerWrapper pipelines used on the initial document. In order to better parallelize the implementation and reduce memory consumption of the resulting document one could try to merge multiple Docker environments as long as the used environments do not conflict with each other. An example of using multiple dockerfiles for a container set up can be found on the docker homepage.

6.2. Cluster viability

One of the main questions remaining are the ability to upscale the presented solution on a server cluster. The common usage of large transformer models in NLP highlights the need of NLP processes to have access to a lot of computation power. Showing the viability of the presented solution in a cluster environment with the use of apache UIMA Ducc or Kubernetes is one of the main missing points of the thesis.

6.3. Storage pressure reduction

At the moment the implementation takes up a lot of space, a common pipeline on the corpus of English Wikipedia Texts will increase the size of the created CAS collection by multiple GB. This stems from the problem that every CAS document contains all information to reproduce every step in the pipelines. In order to improve the storage consumption one could store only an address/hash to the used data for the analysis engine. This would lead to an impressive size reduction since all analysis engine information is only stored once. Building on the common problem that internet endpoint availability tends to detoriate over time, one could build a system of trust around the information stored in the internet. Based on the current interest in technologies like blockchains one could introduce a blockchain which saves the configuration of NLP pipelines and the consensus algorithm, a variant of the common proof of work algorithm, could proof the work by validating prior analysis engine configurations.

6.4. Cryptographic safety

At the moment the integrity of the CAS is documented by storing a CRC32 of all annotations made in the wrapped analysis engine. This system can be fooled with ease. One can add as many annotations as one likes and adds at the end a new CRC32 hash to the document. The implementation can not distinguish this annotation from self made annotations. Another direction of research would be to implement a fool proof system which cryptographically connects the annotations made with the engine which is run on the CAS document.

Glossary

AAAI The AAAI Conference on Artificial Intelligence, https://aaai.org/Conferences/AAAI/aaai.php. 6, 10

AWS Amazon web services, the largest provider of cloud based infrastructure. source. 12, 17, 32

DockerWrapper The provided implementation which wraps an arbitary UIMA analysis engine inside a containerized environment.. 9, 23, 24, 27, 28, 29, 30, 31, 33, 35, 48

EMNLP The Conference on Empirical Methods in Natural Language Processing. 6, 10

GUI Graphical User Interface. 14

LREC The International Conference on Language Resources and Evaluation, http://www.lrec-conf.org/. 6

NeurIPS Conference on Neural Information Processing Systems, https://nips.cc/. 11

NLP Natural language processing, the field of computer science concerned with analysing and syntheising textual data.. 3, 6, 7, 8, 11, 15, 19, 28, 35

SoFa Subject OF Analysis of the apache uima specification. For more information see link.. 20

UIMA Unstructured information management application. 6

VIM International Vocabulary of Metrology (JCGM, 2012). 7

WMS Workflow Management System. 15

Bibliography

Giuseppe Abrami. Unleashing annotations with TextAnnotator: Multimedia, multi-perspective document views for ubiquitous annotation Wikidition View project Alignment in Communication View project. (June), 2021. URL `https://www.researchgate.net/publication/351345638`.

Giuseppe Abrami, Manuel Stoeckel, and Alexander Mehler. Textannotator: A UIMA based tool for the simultaneous and collaborative annotation of texts. *LREC 2020 - 12th International Conference on Language Resources and Evaluation, Conference Proceedings*, (May):891–900, 2020.

Khudran Alzhrani and Amer Aljaedi. Windows and Linux Random Number Generation Process: A Comparative Analysis. *International Journal of Computer Applications*, 113(8):17–25, 2015. doi: 10.5120/19847-1710.

Charles Anderson. Docker. *IEEE Software*, 32(3):102–105, 2015. ISSN 07407459. doi: 10.1109/MS.2015.62.

Apache Software Foundation. openNLP Natural Language Processing Library, 2014. URL `http://opennlp.apache.org/`. http://opennlp.apache.org/.

apache UIMA. Getting started: Working with pears. URL `https://uima.apache.org/doc-uima-pears.html`.

Monya Baker and Dan Penny. Is there a reproducibility crisis? *Nature*, 533(7604): 452–454, 2016. ISSN 14764687. doi: 10.1038/533452A.

Anya Belz, Shubham Agarwal, Anastasia Shimorina, and Ehud Reiter. A systematic review of reproducibility research in natural language processing. *EACL 2021 - 16th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference*, (Ml):381–393, 2021. doi: 10.18653/v1/2021.eacl-main.29.

Jayanti Bhandari Neupane, Ram P Neupane, Yuheng Luo, Wesley Y Yoshida, Rui Sun, and Philip G Williams. Characterization of Leptazolines A–D, Polar Oxazolines from the Cyanobacterium Leptolyngbya sp., Reveals a Glitch with the “Willoughby–Hoye” Scripts for Calculating NMR Chemical Shifts. *Organic Letters*, 21(20):8449–8453, oct 2019. ISSN 1523-7060. doi: 10.1021/acs.orglett.9b03216. URL `https://doi.org/10.1021/acs.orglett.9b03216`.

S T Bhosale, Tejaswini Patil, and Pooja Patil. SQLite: Light Database System. *International Journal of Computer Science and Mobile Computing*, 44(4):882–885, 2015.

Carl Boettiger. An introduction to Docker for reproducible research, with examples from the R environment. 2014. doi: 10.1145/2723872.2723882. URL `http://arxiv.org/abs/1410.0846%0Ahttp://dx.doi.org/10.1145/2723872.2723882`.

Carl Boettiger and Dirk Eddelbuettel. An introduction to Rocker: Docker containers for R. *R Journal*, 9(2):527–536, 2017. ISSN 20734859. doi: 10.32614/rj-2017-065.

António Branco, Nicoletta Calzolari, Piek Vossen, Gertjan van Noord, Dieter van Uytvank, João Silva, Luís Gomes, André Moreira, and Willem Elbers. A shared task of a new, collaborative type to foster reproducibility: A first exercise in the area of language science and technology with REPROLANG2020. *LREC 2020 - 12th International Conference on Language Resources and Evaluation, Conference Proceedings*, (May):5539–5545, 2020.

Jürgen Cito and Harald C. Gall. Using docker containers to improve reproducibility in software engineering research. *Proceedings - International Conference on Software Engineering*, 1:906–907, 2016. ISSN 02705257. doi: 10.1145/2889160.2891057.

Christian Collberg, Todd Proebsting, Gina Moraila, Akash Shankaran, Zuoming Shi, and Alex M Warren. Measuring reproducibility in computer systems research. *Department of Computer Science, University of Arizona, Tech. Rep*, pages 1–37, 2014.

Richard Eckart de Castilho. Natural Language Processing: Integration of Automatic and Manual Analysisde Castilho, R. E. (2014). Natural Language Processing: Integration of Automatic and Manual Analysis. 205. page 205, 2014.

William Digan, Aurélie Névéol, Antoine Neuraz, Maxime Wack, David Baudoin, Anita Burgun, and Bastien Rance. Can reproducibility be improved in clinical natural language processing? A study of 7 clinical NLP suites. *Journal of the American Medical Informatics Association*, 28(3):504–515, 2021. ISSN 1527974X. doi: 10.1093/jamia/ocaa261.

Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, and Noah A. Smith. Show your work: Improved reporting of experimental results. *EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference*, (2):2185–2194, 2020. doi: 10.18653/v1/d19-1224.

Dr. Drummond and Chris Drummond. Replicability is not Reproducibility: Nor is it Good Science. (2005):2005–2008, 2009.

Richard Eckart de Castilho and Iryna Gurevych. A broad-coverage collection of portable NLP components for building shareable analysis pipelines. 2(1):1–11, 2015. doi: 10.3115/v1/w14-5201.

Mark Granroth-Wilding. Pimlico: A toolkit for corpus-processing pipelines and reproducible experiments. pages 101–109, 2020. doi: 10.18653/v1/2020.nlposs-1.14.

Udo Hahn, Franz Matthies, Erik Faessler, and Johannes Hellrich. UIMA-based JCoRe 2.0 goes GitHub and Maven Central - State-of-the-art software resource engineering and distribution of NLP pipelines. *Proceedings of the 10th International Conference on Language Resources and Evaluation, LREC 2016*, pages 2502–2509, 2016.

Wahed Hemati, Tolga Uslu, and Alexander Mehler. TextImager: A distributed UIMA-based system for NLP. *COLING 2016 - 26th International Conference on Computational Linguistics, Proceedings of COLING 2016: System Demonstrations*, pages 59–63, 2016.

Tobias Horsmann and Torsten Zesch. DeepTC - An extension of DKPro text classification for fostering reproducibility of deep learning experiments. *LREC 2018 - 11th International Conference on Language Resources and Evaluation*, pages 2539–2545, 2019.

JCGM. *International vocabulary of metrology – Basic and general concepts and associated terms (VIM).* 2012. doi: 10.1016/j.tetlet.2017.07.069. URL `https://www.bipm.org/documents/20126/2071204/JCGM_200_2012.pdf/f0e1ad45-d337-bbeb-53a6-15fe649d0ff1`.

Austin Jun-Yian. Overcoming Code Rot in Legacy Software Projects. (2016), 2017.

Jan-Christoph Klie and Richard Eckart de Castilho. Dkpro cassis - reading and writing uima cas files in python. URL `https://github.com/dkpro/dkpro-cassis`.

Fachbereich Mathematik. With UTEX into the Nineties. 10(4):681–690, 1989.

Prajval Mohan, Tejas Jambhale, Lakshya Sharma, Simran Koul, and Simriti Koul. Load Balancing using Docker and Kubernetes: A Comparative Study. *International Journal of Recent Technology and Engineering*, 9(2):782–792, 2020. doi: 10.35940/ijrte.b3938.079220.

Mark Neumann, Daniel King, Iz Beltagy, and Waleed Ammar. ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing. 0:319–327, 2019. ISSN 2331-8422. doi: 10.18653/v1/w19-5034.

Carl V. Phillips. Publication bias in situ. *BMC Medical Research Methodology*, 4:1–11, 2004. ISSN 14712288. doi: 10.1186/1471-2288-4-20.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A python natural language processing toolkit for many human languages, 2020.

Nicolas P. Rougier, Konrad Hinsen, Frédéric Alexandre, Thomas Arildsen, Lorena A. Barba, Abien C.Y.Benureau, C. Titus Brown, Pierre DeBuy, Ozan Caglayan, Andrew P. Davison, Marc André Delsuc, Georgios Detorakis, Alexandra K. Diem, Damien Drix, Pierre Enel, Benoît Girard, Olivia Guest, Matt G. Hall, Rafael N. Henriques, Xavier Hinaut, Kamil S. Jaron, Mehdi Khamassi, Almar Klein, Tiina Manninen, Pietro Marchesi, Daniel McGlinn, Christoph Metzner, Owen Petchey, Hans Ekkehard Plesser, Timothée Poisot, Karthik Ram, Yoav Ram, Etienne Roesch, Cyrille Rossant, Vahid Rostami, Aaron Shifman, Joseph Stachelek, Marcel Stimberg, Frank Stollmeier, Federico Vaggi, Guillaume Viejo, Julien Vitay, Anya E. Vostinar, Roman Yurchak, and Tiziano Zito. Sustainable computational science: The ReScience Initiative. *PeerJ Computer Science*, 2017(12):1–17, 2017. ISSN 23765992. doi: 10.7717/peerj-cs.142.

Guergana K. Savova, James J. Masanz, Philip V. Ogren, Jiaping Zheng, Sunghwan Sohn, Karin C. Kipper-Schuler, and Christopher G. Chute. Mayo clinical Text Analysis and Knowledge Extraction System (cTAKES): Architecture, component evaluation and applications. *Journal of the American Medical Informatics Association*, 17(5):507–513, 2010. ISSN 10675027. doi: 10.1136/jamia.2009.001560.

Matthew Stewart. The Management Myth, 2006. URL `https://www.theatlantic.com/magazine/archive/2006/06/the-management-myth/304883/`.

Apache UIMA. Uima framework core. URL `https://uima.apache.org/doc-uimacpp-huh.html`.

Kirstie Whitaker. The Machine Learning Reproducibility Checklist. Technical report, 2020. URL `https://www.cs.mcgill.ca/$\sim$jpineau/ReproducibilityChecklist.pdf`.

Chen Zhaojun. Apache log4j2 jndi features do not protect against attacker controlled ldap and other jndi related endpoints., 2021. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228`.

A. Appendix

A.1. Code reference

This section provides the code for the extraction programs used to extract the document text from the supplied corpora. It is included in this thesis so that the reader can get a quick grasp on the techniques to extract the documents.

```
cargo run --release -- 20000
```

Listing A.1: The bash code to create the wikipedia sample corpus given the downloaded wikipedia json saved as data.json in the root directory

```
#![feature(iter_advance_by)]
use std::fs::File;
use std::io::{self, prelude::*, BufReader};
use rand::prelude::*;
use std::collections::HashSet;
use std::vec::Vec;
use std::convert::TryInto;
use serde::{Serialize, Deserialize};
use serde_json;
use statrs::statistics::Statistics;

#[derive(Deserialize)]
struct JsonEntry {
    body: String,
}

#[derive(Serialize,Deserialize)]
struct JsonFullFile {
    data: Vec<f64>
}

fn main() -> io::Result<()> {
    let args: Vec<String> = std::env::args().collect();
    if args.len() < 2 {
        panic!("Please add a sample size for the command line usage!");
    }
    let sample_size: usize = args.get(1).unwrap().parse::<usize>().unwrap()
    ;
    let file = File::open("data")?;
    let reader = BufReader::new(file);

    let lines: u64 = reader.lines().count().try_into().unwrap();
    let mut rng = rand::thread_rng();

    let mut next : HashSet<u64> = HashSet::with_capacity(sample_size);
    loop {
        let next_line = rng.next_u64()%lines;
        if next.get(&next_line).is_none() {
            next.insert(next_line);
            if next.len() == sample_size {
                break;
            }
        }
```

```rust
    }
    println!("{}",next.len());

    let file2 = File::open("data")?;
    let reader2 = BufReader::new(file2);

    println!("Fetched the rows of data, serializing now to disk!");

    let mut counter = 0;
    let mut full_vec : Vec<f64> = Vec::new();
    let mut sample_vec: Vec<f64> = Vec::new();

    for x in reader2.lines() {
        let value : JsonEntry = serde_json::from_str(&x.unwrap()).unwrap();
        if value.body.len() != 0 {
            full_vec.push(value.body.len() as f64);
        }
        if next.contains(&counter) {
            if value.body.len() != 0 && sample_vec.len() < sample_size {
                sample_vec.push(value.body.len() as f64);
                std::fs::write(String::from("output/")+&counter.to_string()
    +".txt", value.body).unwrap();
            }
            else if value.body.len() == 0 {
                let mut next_elem = counter+1;
                loop {
                    if next.contains(&next_elem) {
                        next_elem+=1;
                    }
                    else {
                        break;
                    }
                }
                next.insert(next_elem);
            }
        }
        counter+=1;
    }

    println!("Sample mean {} and variance {}",(&sample_vec).mean(), (&
    sample_vec).variance());
    println!("Full corpus mean {} and variance {}",(&full_vec).mean(), (&
    full_vec).variance());
    let val = JsonFullFile{ data: full_vec };
    std::fs::write("output/full_text_lengths.json",serde_json::to_string(&
    val).unwrap()).unwrap();
    let val_sample = JsonFullFile{ data: sample_vec };
    std::fs::write("output/sample_text_lengths.json",serde_json::to_string
    (&val_sample).unwrap()).unwrap();

    Ok(())
}
```

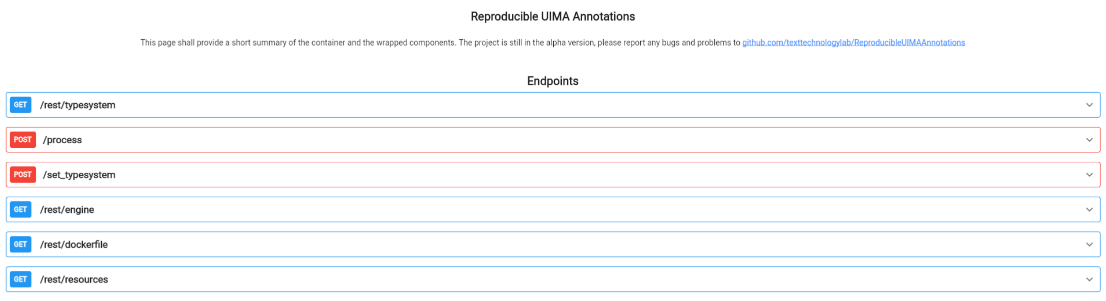
Listing A.2: The rust code to create the sample of the wikipedia corpus from a JSON wikipedia dump. Further information can be found in the github project.

```python
import xml.etree.ElementTree as ET
import sys
import os

def dump(iterator,counter):
    for text in iterator:
        counter+=1
        with open('output/'+str(counter)+'.txt','w') as f:
            if text.text != None:
                f.write(text.text)
    return counter



os.mkdir("output")
root = ET.parse('./Ausw\"artigesAmt.xml').getroot()
root2 = ET.parse('./Bundesregierung.xml').getroot()
root3 = ET.parse('./Bundespr\"asidenten.xml').getroot()
root4 = ET.parse('./Bundestagspr\"asidenten.xml').getroot()

counter = 0
counter = dump(root.iter("rohtext"),counter)
counter = dump(root2.iter("rohtext"),counter)
counter = dump(root3.iter("rohtext"),counter)
counter = dump(root4.iter("rohtext"),counter)
```

Listing A.3: The python code to extract the text from the German Political Speeches 2019 Corpus. Further information can be found in the github project.

A.2. UI reference

This section provides a few more screenshots to give a rough overview of the look and feel of the UI.



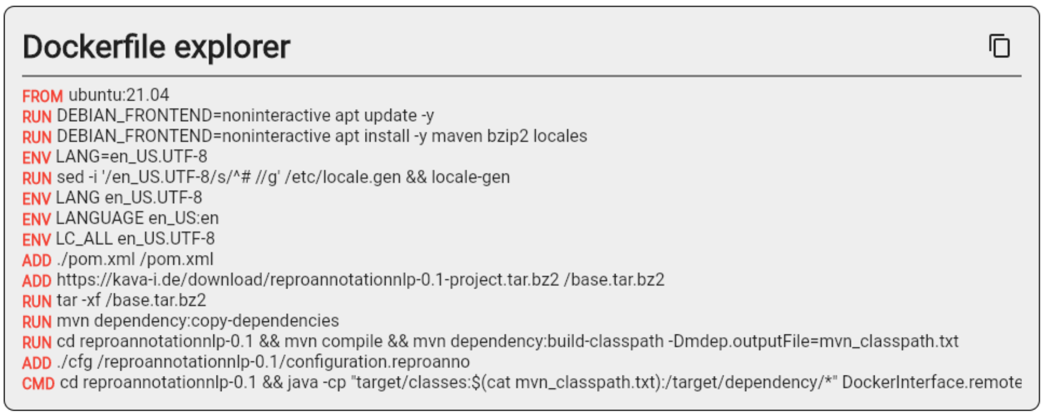
Reproducible UIMA Annotations

This page shall provide a short summary of the container and the wrapped components. The project is still in the alpha version, please report any bugs and problems to github.com/texttechnologylab/ReproducibleUIMAAnnotations

Endpoints

GET /rest/typesystem
POST /process
POST /set_typesystem
GET /rest/engine
GET /rest/dockerfile
GET /rest/resources

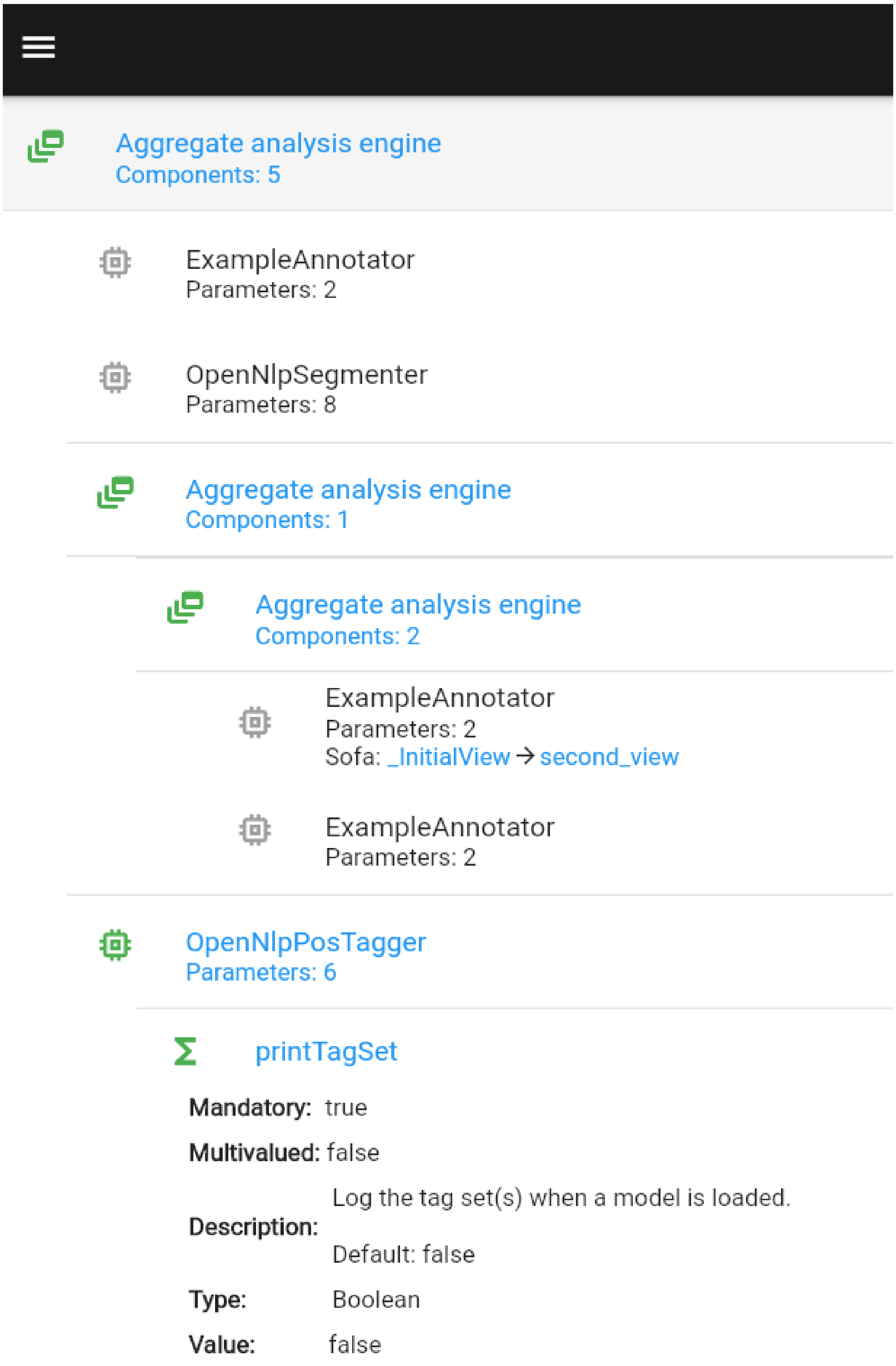
Figure A.1.: The documentation main screen of every created container



Dockerfile explorer

```
FROM ubuntu:21.04
RUN DEBIAN_FRONTEND=noninteractive apt update -y
RUN DEBIAN_FRONTEND=noninteractive apt install -y maven bzip2 locales
ENV LANG=en_US.UTF-8
RUN sed -i '/en_US.UTF-8/s/^# //g' /etc/locale.gen && locale-gen
ENV LANG en_US.UTF-8
ENV LANGUAGE en_US:en
ENV LC_ALL en_US.UTF-8
ADD ./pom.xml /pom.xml
ADD https://kava-i.de/download/reproannotationnlp-0.1-project.tar.bz2 /base.tar.bz2
RUN tar -xf /base.tar.bz2
RUN mvn dependency:copy-dependencies
RUN cd reproannotationnlp-0.1 && mvn compile && mvn dependency:build-classpath -Dmdep.outputFile=mvn_classpath.txt
ADD ./cfg /reproannotationnlp-0.1/configuration.reproanno
CMD cd reproannotationnlp-0.1 && java -cp "target/classes:$(cat mvn_classpath.txt):/target/dependency/*" DockerInterface.remote
```

Figure A.2.: The dockerfile view of every created container



Aggregate analysis engine
Components: 5

ExampleAnnotator
Parameters: 2

OpenNlpSegmenter
Parameters: 8

Aggregate analysis engine
Components: 1

Aggregate analysis engine
Components: 2

ExampleAnnotator
Parameters: 2
Sofa: _InitialView → second_view

ExampleAnnotator
Parameters: 2

OpenNlpPosTagger
Parameters: 6

printTagSet

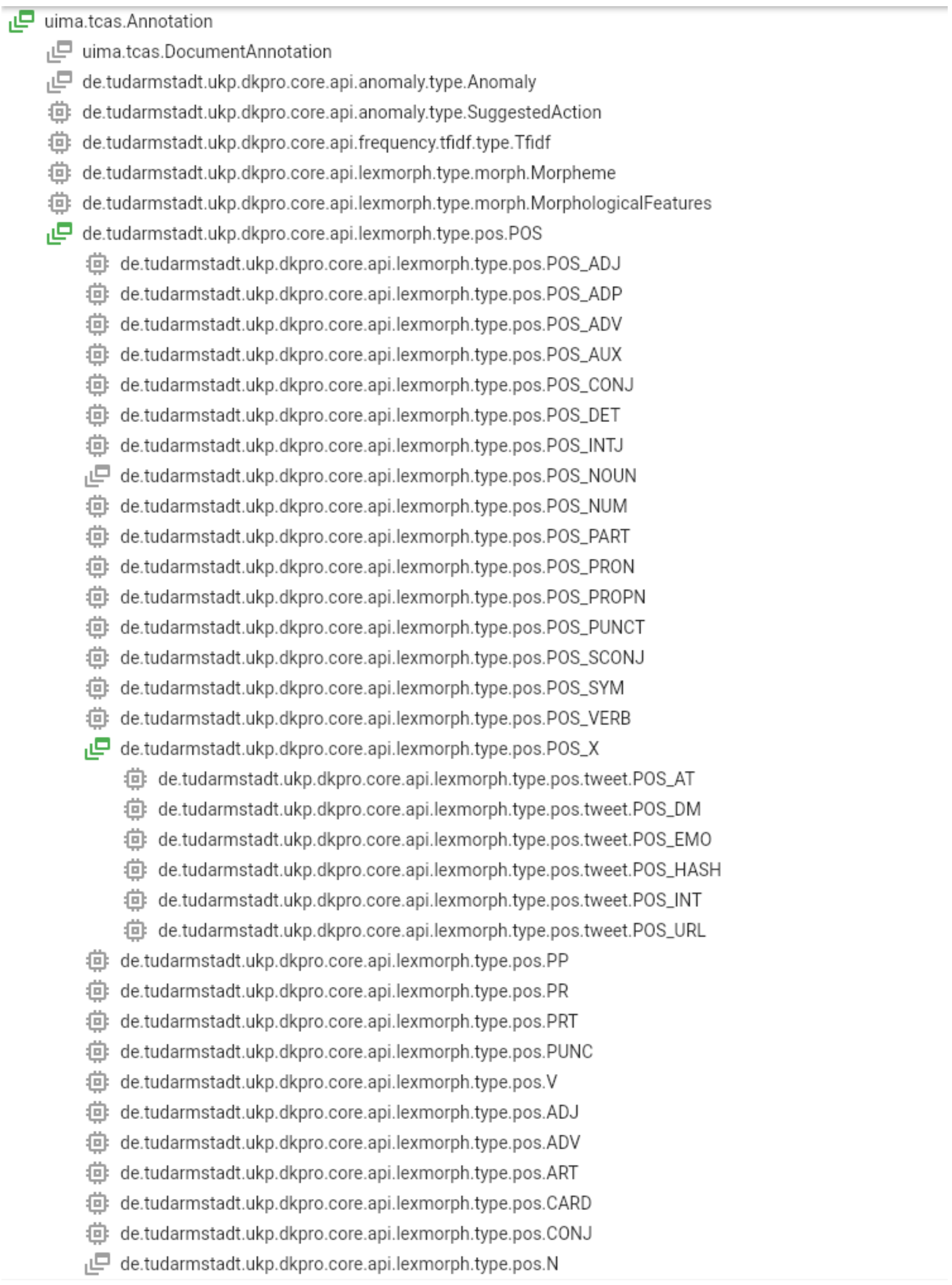
Mandatory:	true
Multivalued:	false
Description:	Log the tag set(s) when a model is loaded. Default: false
Type:	Boolean
Value:	false

Figure A.3.: The analysis engine view of the created container

pom.xml
string

dockerfile
string

Figure A.4.: The resources view of the created container



```
uima.tcas.Annotation
  uima.tcas.DocumentAnnotation
  de.tudarmstadt.ukp.dkpro.core.api.anomaly.type.Anomaly
  de.tudarmstadt.ukp.dkpro.core.api.anomaly.type.SuggestedAction
  de.tudarmstadt.ukp.dkpro.core.api.frequency.tfidf.type.Tfidf
  de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.morph.Morpheme
  de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.morph.MorphologicalFeatures
  de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_ADJ
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_ADP
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_ADV
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_AUX
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_CONJ
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_DET
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_INTJ
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_NOUN
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_NUM
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_PART
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_PRON
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_PROPN
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_PUNCT
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_SCONJ
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_SYM
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_VERB
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS_X
      de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.tweet.POS_AT
      de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.tweet.POS_DM
      de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.tweet.POS_EMO
      de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.tweet.POS_HASH
      de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.tweet.POS_INT
      de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.tweet.POS_URL
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.PP
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.PR
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.PRT
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.PUNC
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.V
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.ADJ
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.ADV
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.ART
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.CARD
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.CONJ
    de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.N
```

Figure A.5.: The typesystem view of the created container

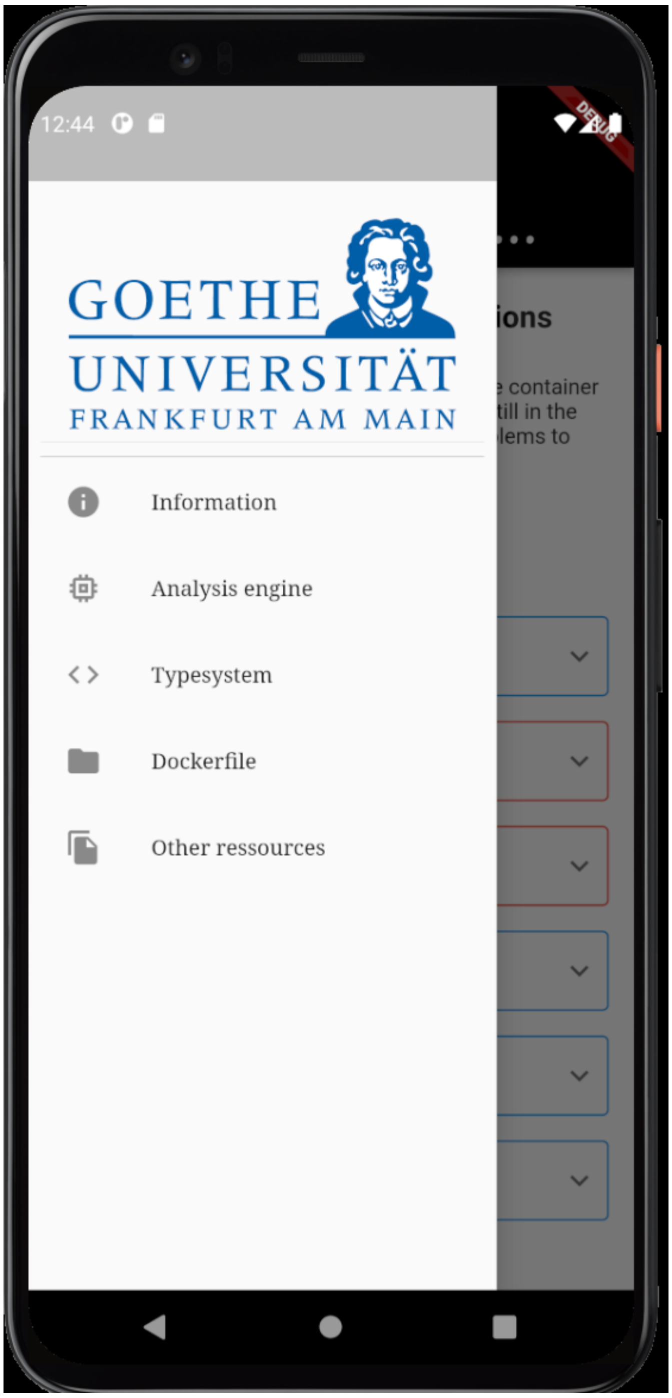
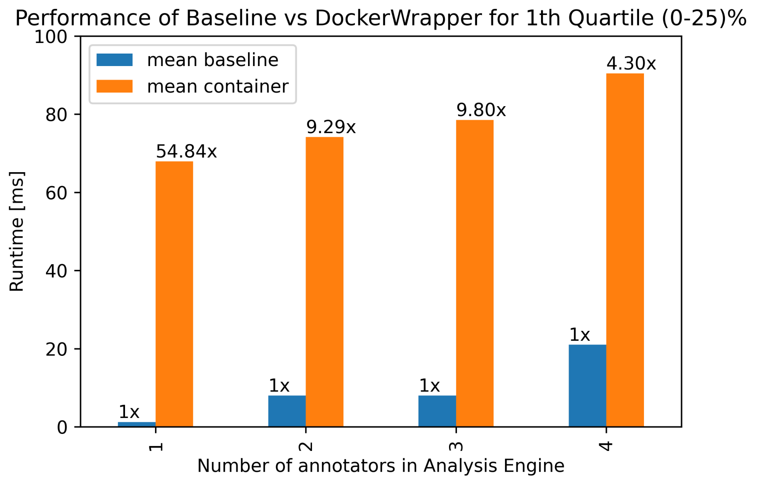


Figure A.6.: An example of the UI on a mobile device.

A.3. Additional figures

This section provides additional figures which would have disrupted the text flow in the main document. They can be used as a reference for additional calculations that have been made but were not included.



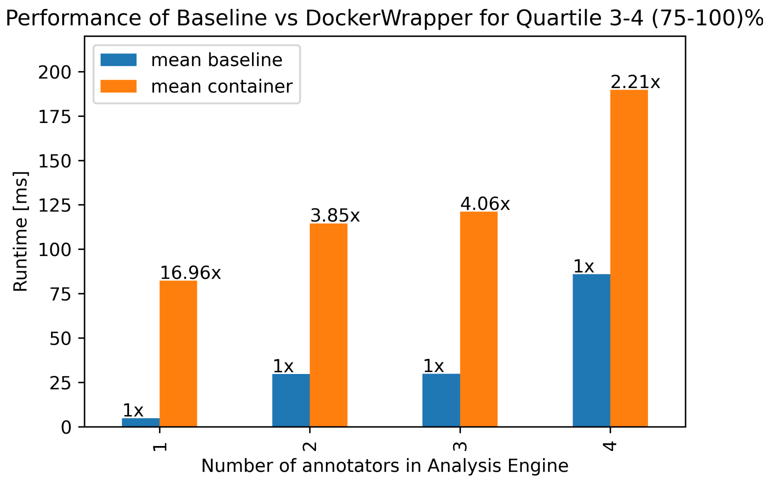


Figure A.7.: The performance of the containerized DockerWrapper vs Baseline on the german political speeches corpus based on size quartiles.