Natural Language Processing with UIMA and DKPro

Tristan Miller

TECHNISCHE UNIVERSITÄT DARMSTADT

Presented at:
School of Data Analysis and Artificial Intelligence
National Research University – Higher School of Economics
22 May 2017

Tristan Miller

Ubiquitous Knowledge Processing Lab
Technische Universität Darmstadt

- Postdoctoral researcher at UKP
- Free software developer
- Science popularizer
- DKPro contributor

https://logological.org

logological

logological

Technische Universität Darmstadt

Ubiquitous Knowledge Processing Lab

Prof. Iryna Gurevych
Technische Universität Darmstadt

- Argumentation mining
- Language technology for the digital humanities
- Lexical-semantic resources and algorithms
- Text mining and analytics
- Writing assistance and language learning

https://www.ukp.tu-darmstadt.de/

UKPLab

University of Regina

University of Toronto

Babel: The Language Magazine

http://babelzine.com

Agenda

TECHNISCHE UNIVERSITÄT DARMSTADT

- The DKPro ecosystem
- Apache UIMA
- DKPro Core
  - Repository-based approach
  - DKPro Script
- DKPro Core metadata

THE DKPRO ECOSYSTEM

DKPro

- Community of projects
- Facilitates NLP research and teaching
- Portable and interoperable software

- Philosophy
  - Projects have a strong relationship with each other
  - Projects share a common ideology of reusability
  - Projects often build upon each other
  - Open source/free software (ASL, GPL)

DKPro in the classroom

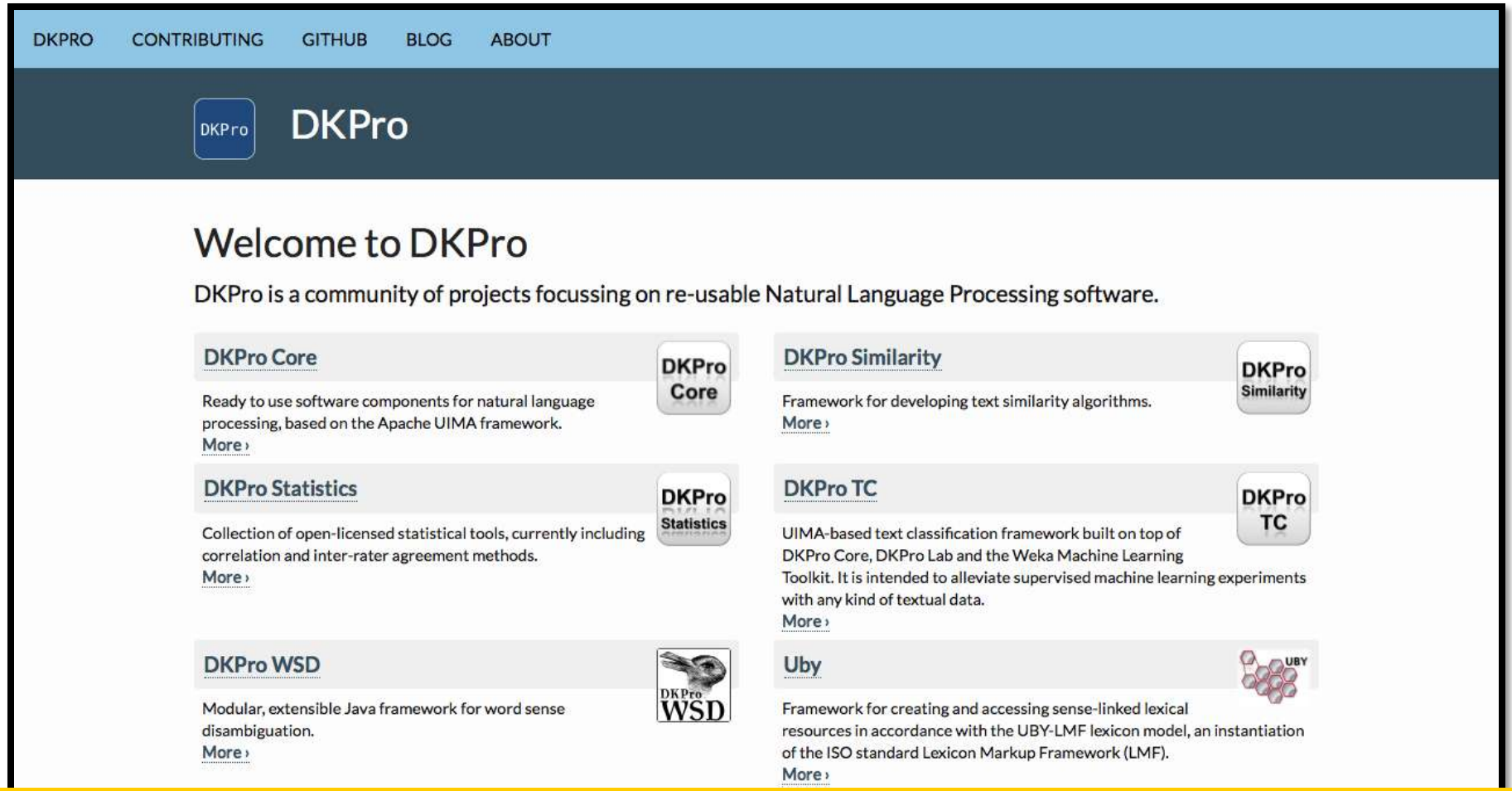
- Reduces the barrier to entry for learning and applying natural language processing
- No need to implement lower-level NLP tasks from scratch
- Component-based architecture can streamline grading of projects

- TU Darmstadt courses using DKPro:
  - Natural Language Processing for the Web
  - Unstructured Information Management
  - Natural Language Processing and eLearning
  - Lexical-semantic Methods for Language Understanding

DKPro

Reusable software for NLP

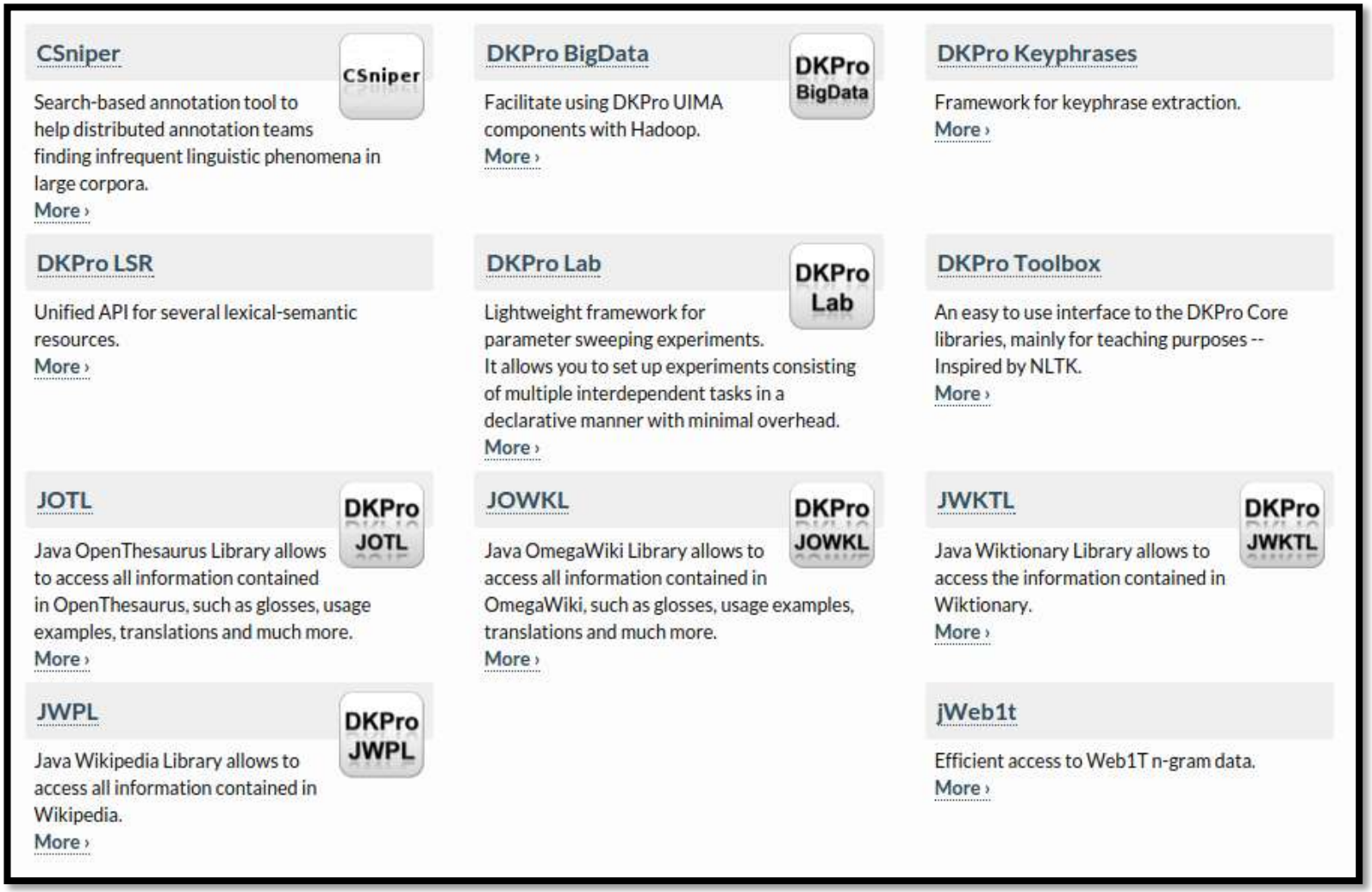
TECHNISCHE UNIVERSITÄT DARMSTADT



https://dkpro.org

DKPro
Reusable software for NLP

TECHNISCHE UNIVERSITÄT DARMSTADT



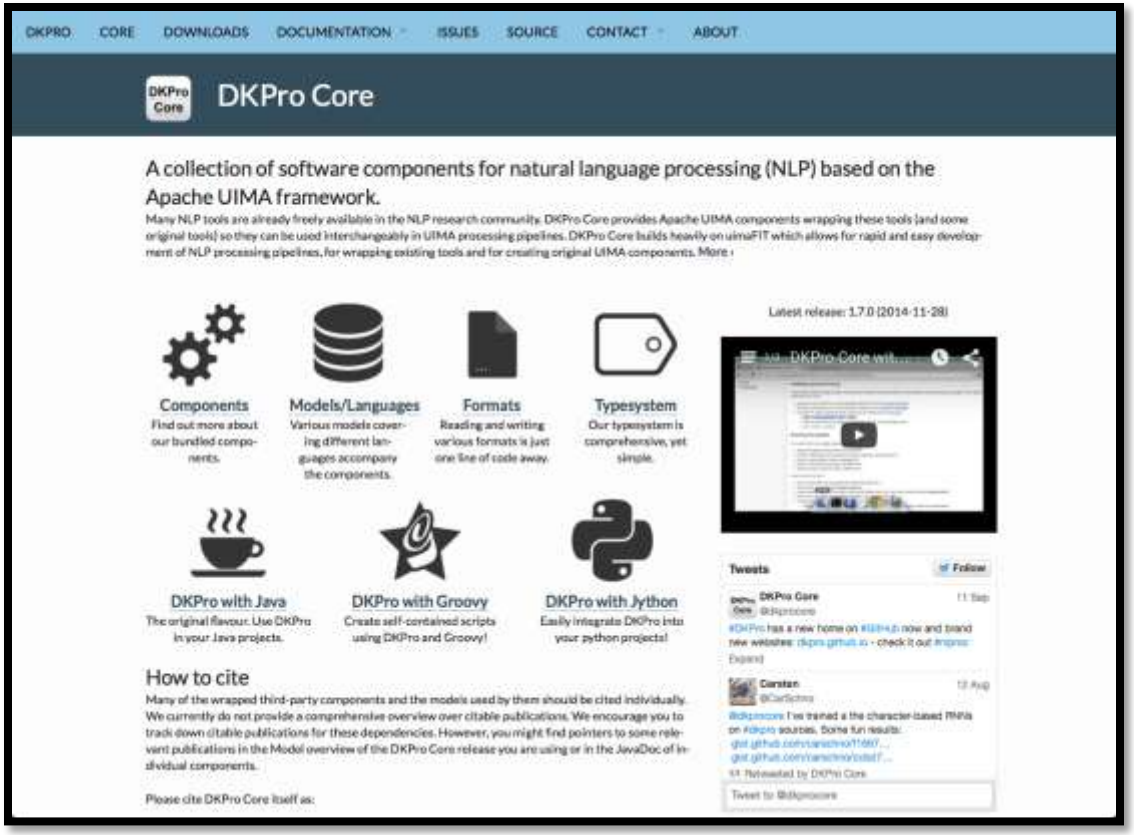
UIMA-based linguistic preprocessing

DKPro Core

- NLP
- Normalization
- Preprocessing for ML
- Mix & match components
- Convert between formats

- Train models (new)
- Evaluate (new)

- Experimental pipelines
- Embed in applications
- Ready to run on server/cluster



https://dkpro.github.io/dkpro-core

Beyond the pipeline…
DKPro Lab

TECHNISCHE UNIVERSITÄT DARMSTADT

Conduct experiments

1. with a **lightweight declarative** set up

2. with **parameter sweeping**

3. in a **reproducible** manner

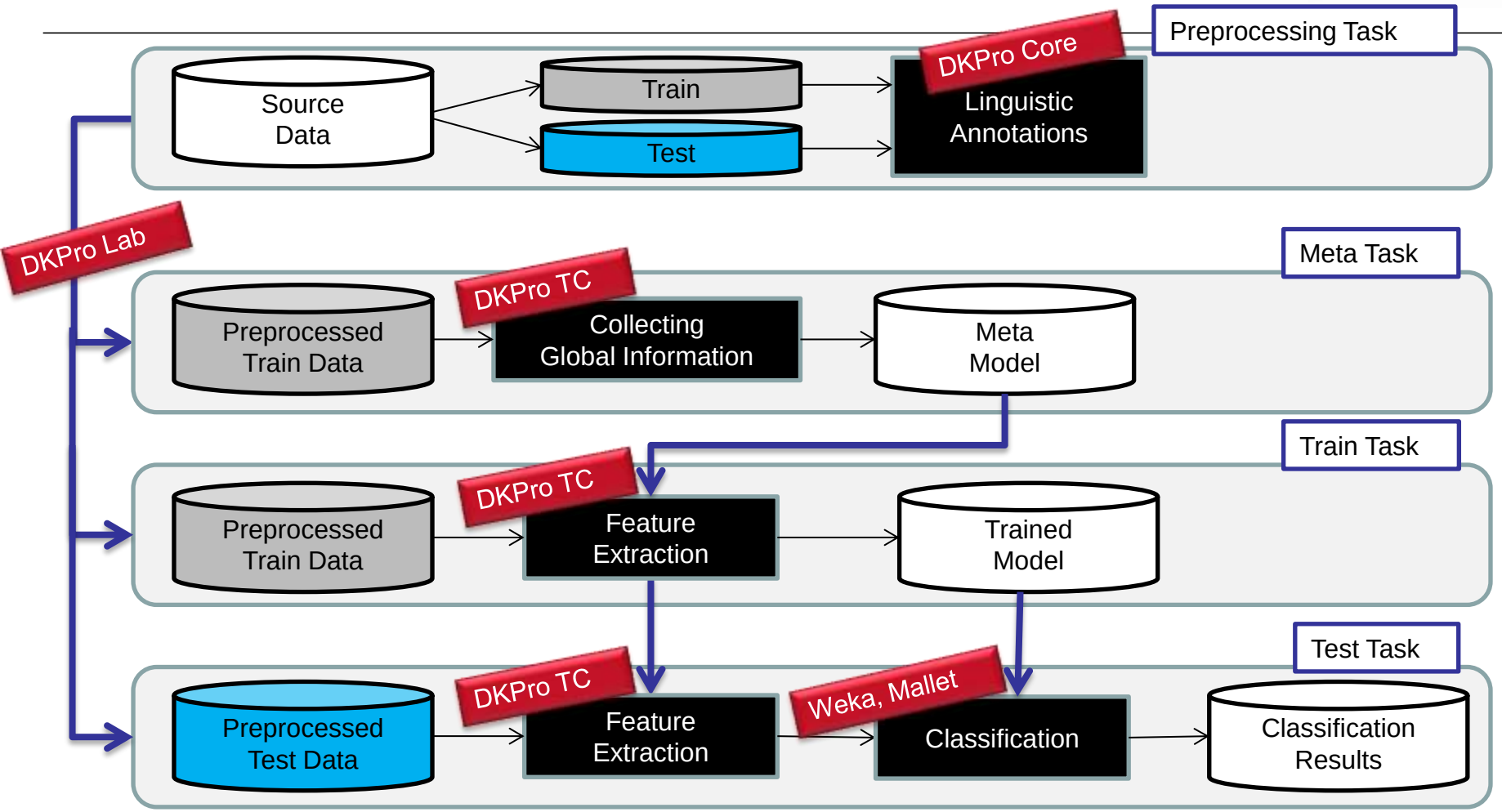
Generic core framework for arbitrary experiments

Extensions for application domains (e.g., ML)

https://dkpro.github.io/dkpro-lab

Experiments with machine learning…

DKPro TC



https://dkpro.github.io/dkpro-tc

DKPro TC
Example: Sentiment Detection on Tweets

TECHNISCHE UNIVERSITÄT DARMSTADT

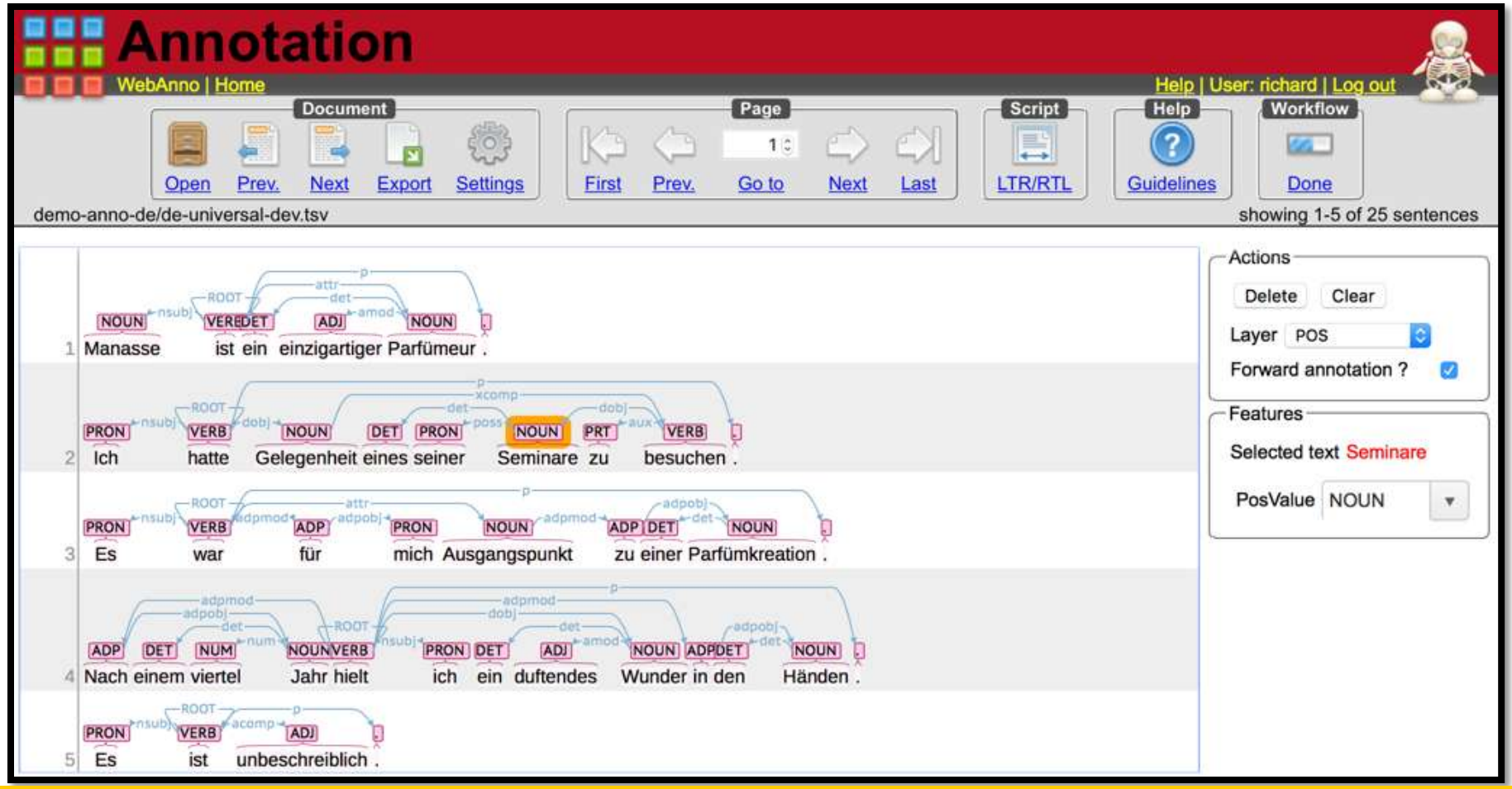
- Set up a parameter space configuration
- Leave the rest to DKPro TC / Lab

Example Experiment Configuration: 10-fold cross-validation

```
BatchTaskCrossValidation batchTask = [
   experimentName: "Twitter Sentiment",
   preprocessingPipeline: createEngineDescription(ArkTweetTagger),
   parameterSpace: [// multi-valued parameters will be swept
      Dimension.createBundle("reader", [
         readerTrain: LabeledTweetReader,
         readerTrainParams: [LabeledTweetReader.PARAM_CORPUS_PATH, "tweets.txt"]]),
      Dimension.create("featureMode", "document"),
      Dimension.create("learningMode", "singleLabel"),
      Dimension.create("featureSet", [EmoticonRatioExtractor.name,
                                      NumberOfHashTagsExtractor.name]),
      Dimension.create("dataWriter", WekaDataWriter.name),
      Dimension.create("classificationArguments", [NaiveBayes.name], [RandomForest.name])],
      reports: [BatchCrossValidationReport], // collects results from folds
      numFolds: 10];
```

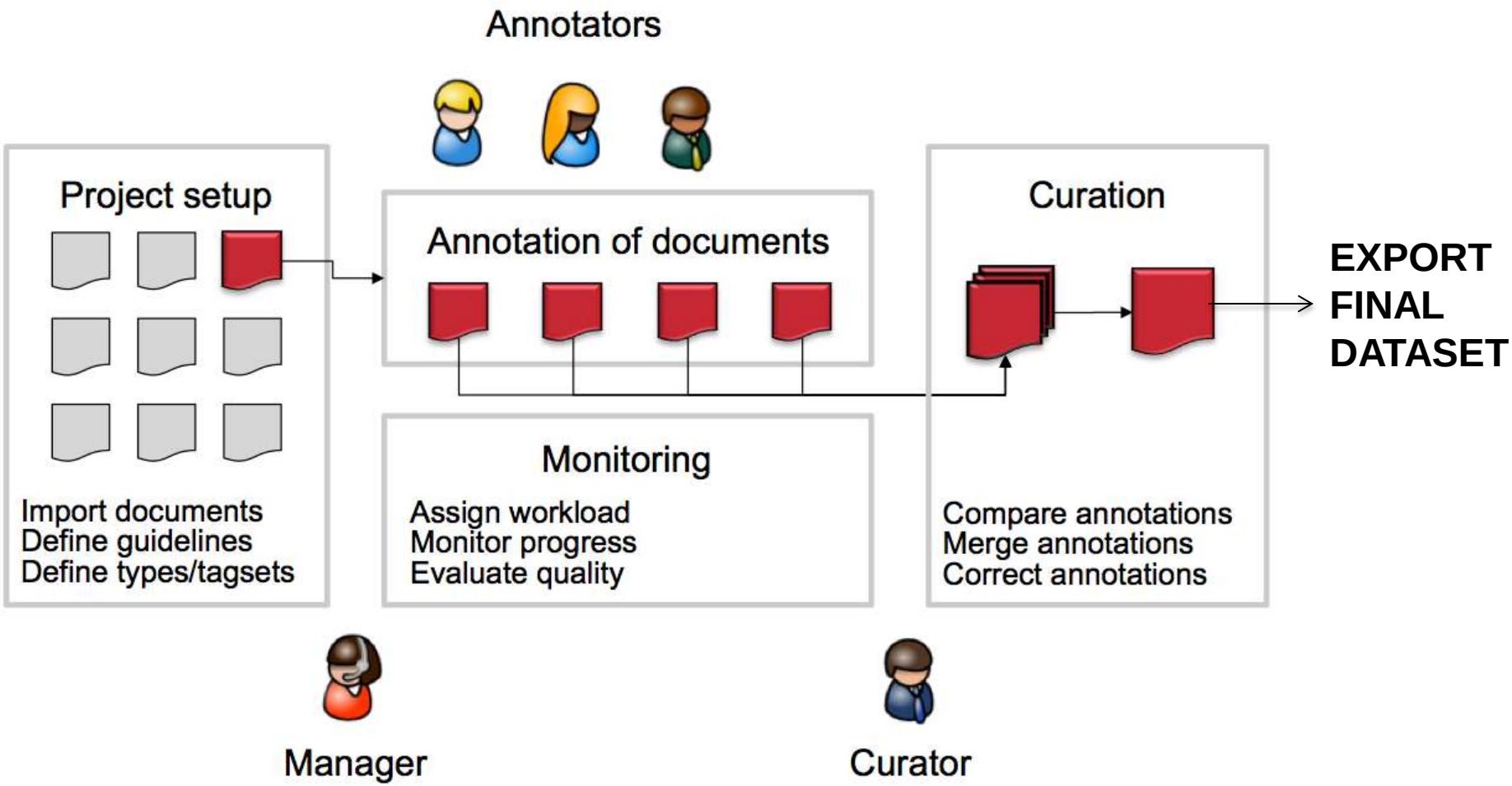
WebAnno

TECHNISCHE UNIVERSITÄT DARMSTADT



https://webanno.github.io/webanno

WebAnno
Workflow



WebAnno
Properties

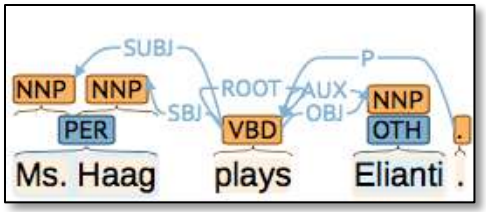
- **Compatible with DKPro Core**
  - Builds on DKPro Core type system
  - Uses DKPro Core components for import/export
- **Flexible**
  - Configurable annotation layers
  - Different annotation modes including correction and automation
- **Web-based**
  - Available to annotators everywhere, no installation effort
  - All configuration performed through the web interface
- **Installable and platform independent**
  - Run your own WebAnno server for your group
  - Use the WebAnno standalone version when working alone
  - Platform independent Java-based server
- **Free/open source software**
  - Allows the community to participate

WebAnno

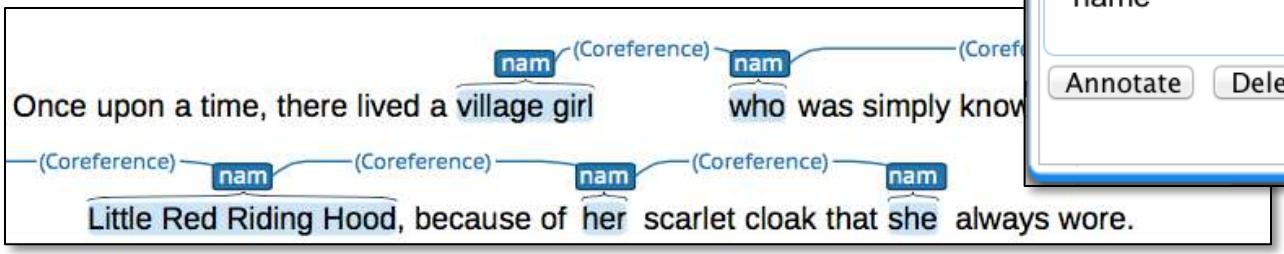
Annotation layer examples

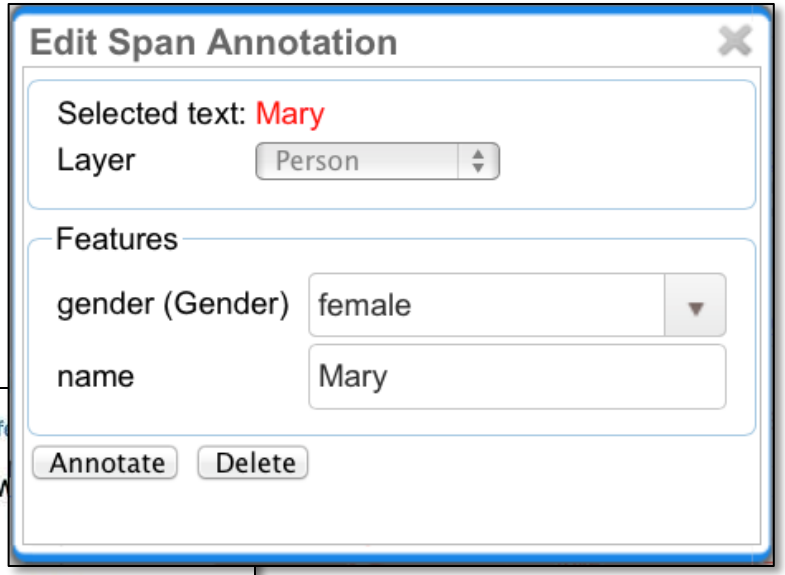
TECHNISCHE UNIVERSITÄT DARMSTADT

Part-of-Speech & Dependency layers

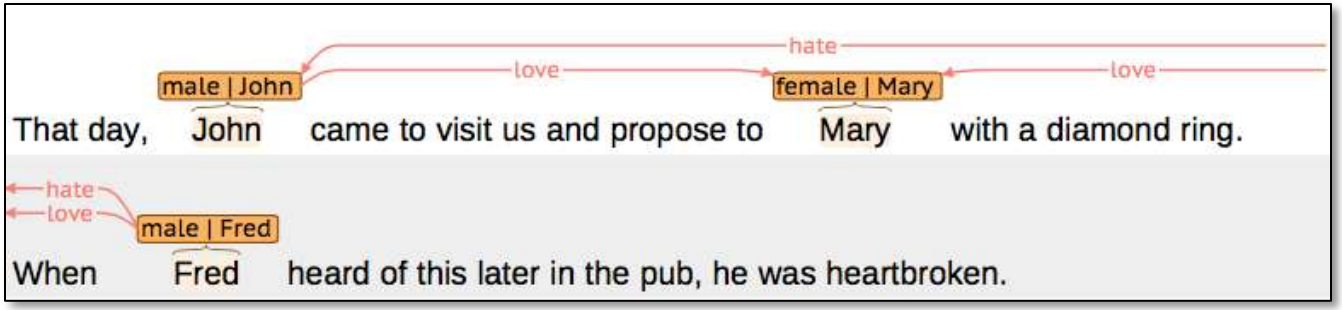


Coreference layer

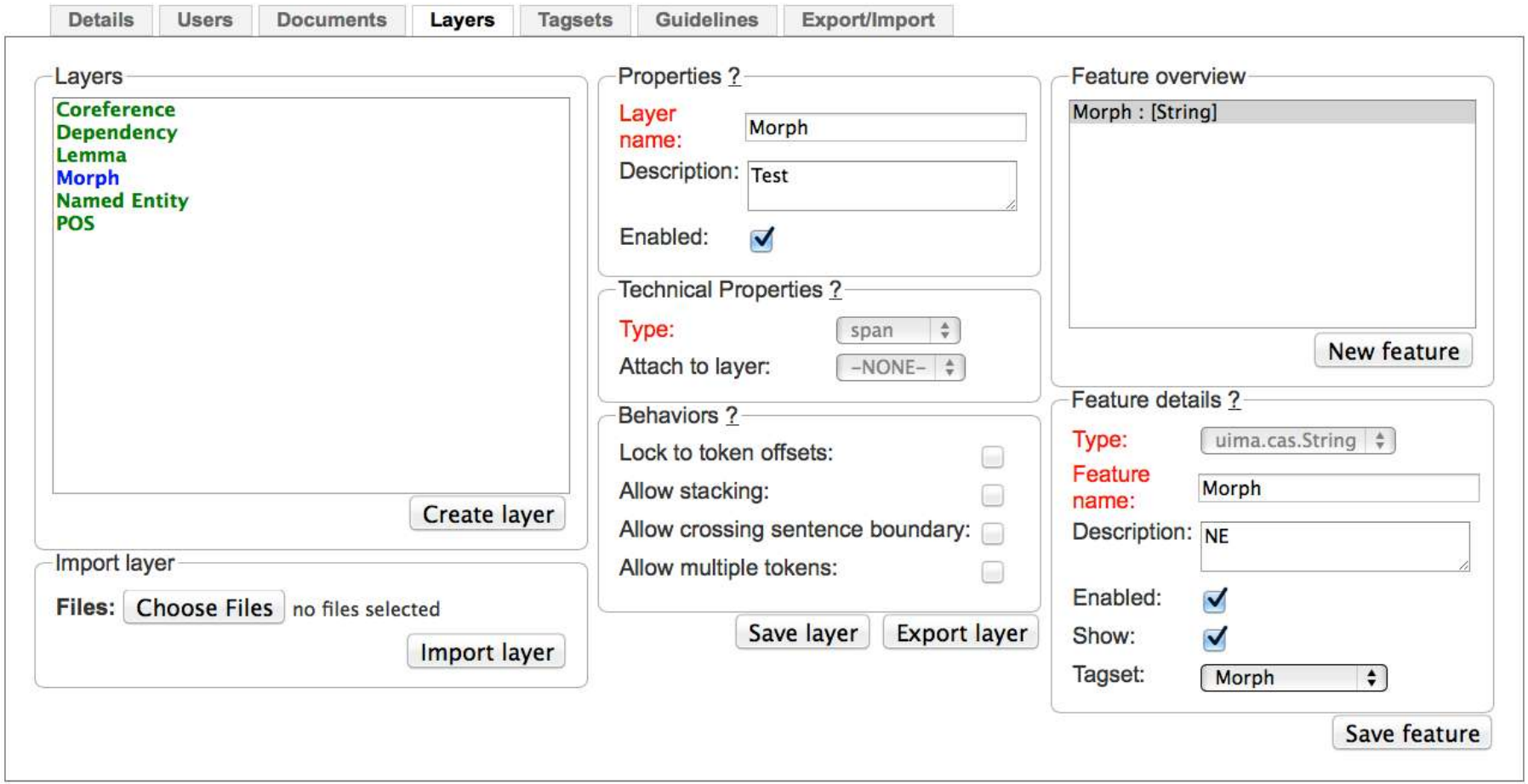




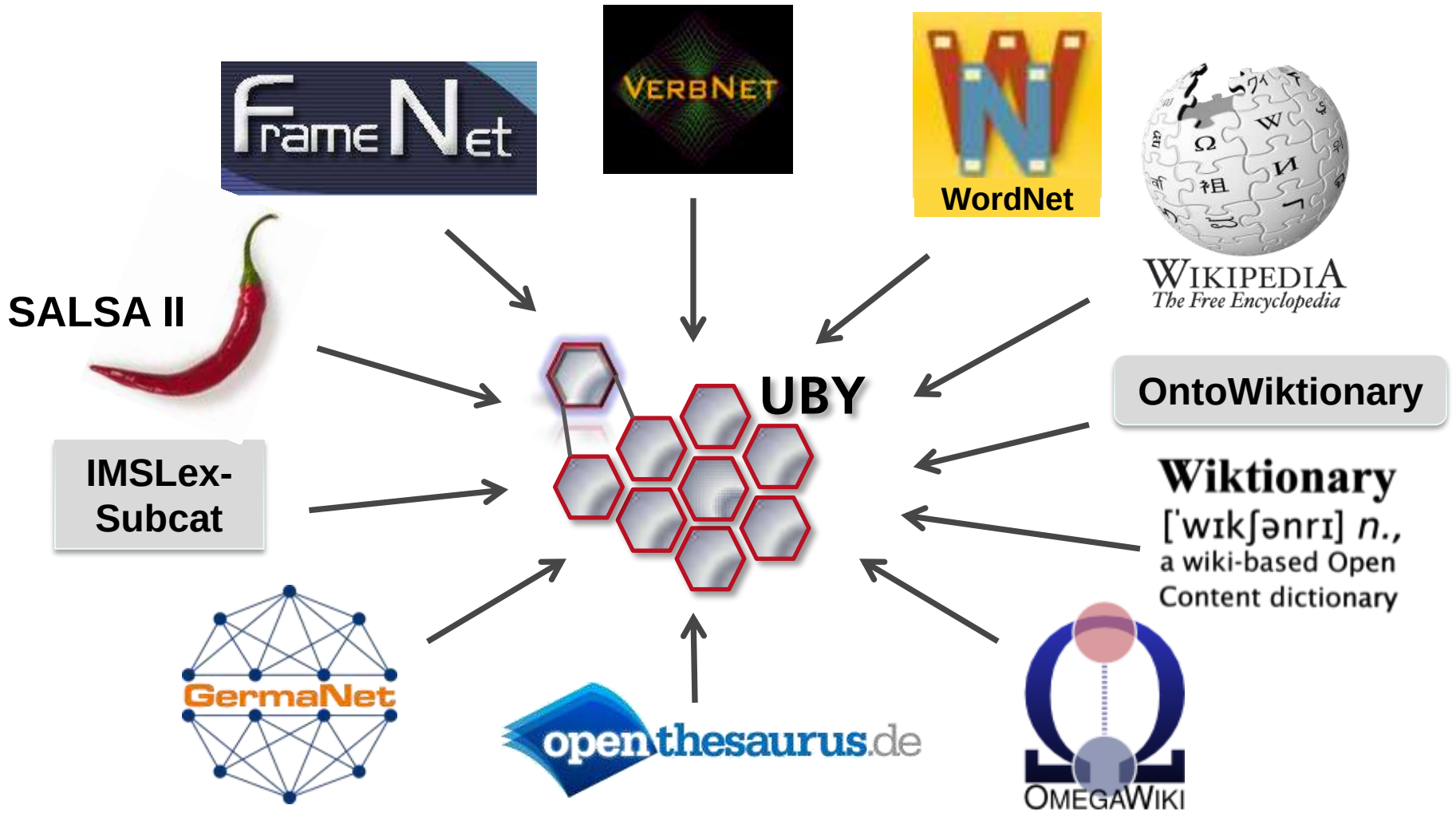
Custom Person (span) / Relationship (relation) layers



WebAnno
Custom annotation layers

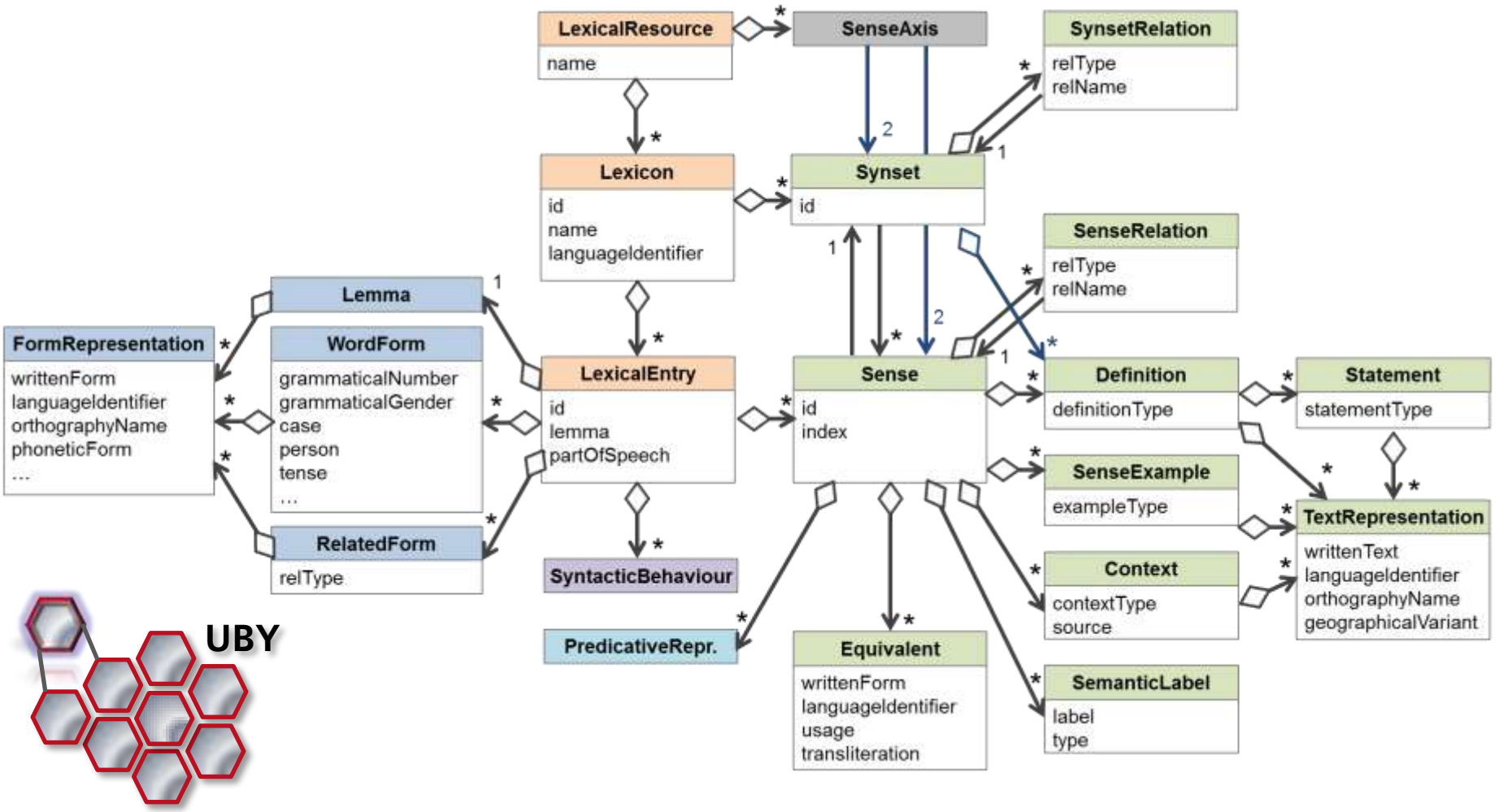


UBY

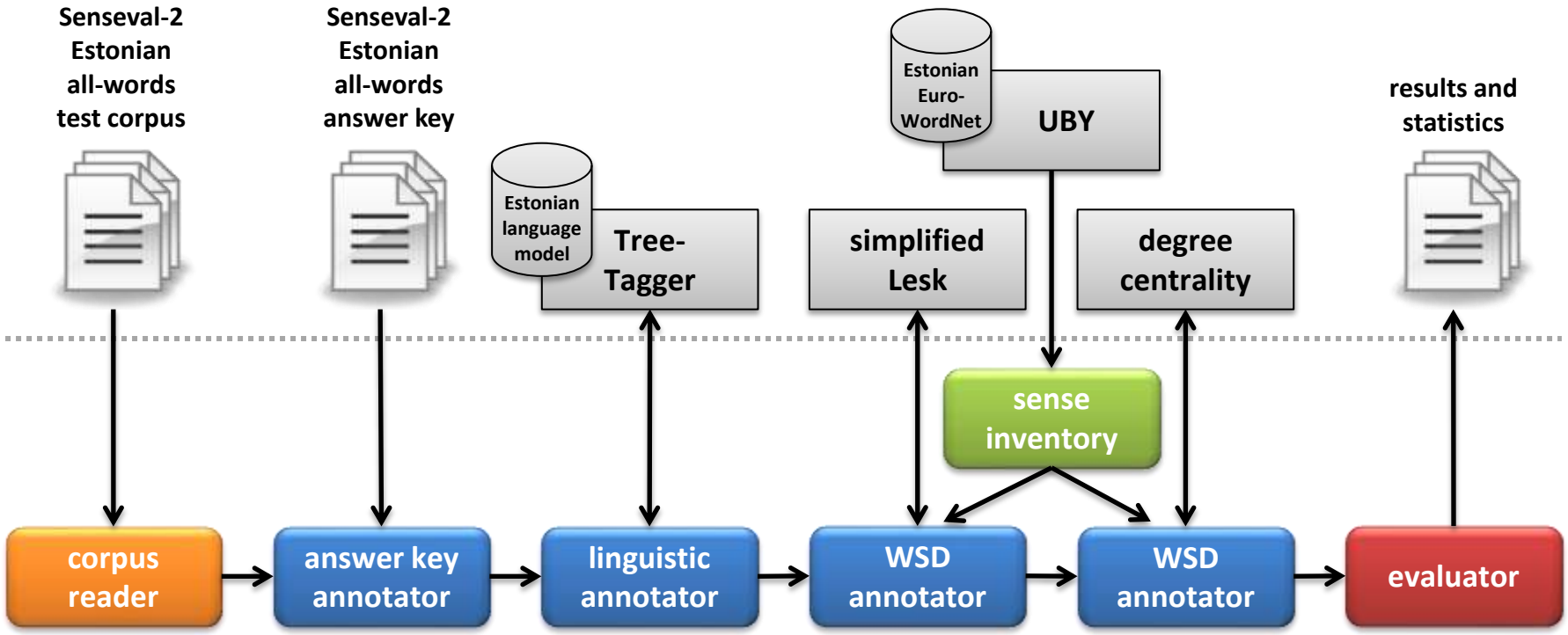

FrameNet
VerbNet
WordNet
Wikipedia
The Free Encyclopedia
SALSA II
UBY
OntoWiktionary
IMSLex-Subcat
Wiktionary
['wɪkʃənrɪ] n.,
a wiki-based Open Content dictionary
GermaNet
openthesaurus.de
OmegaWiki

UBY

TECHNISCHE UNIVERSITÄT DARMSTADT



DKPro WSD


Senseval-2 Estonian all-words test corpus
Senseval-2 Estonian all-words answer key
Estonian language model
Tree-Tagger
Estonian Euro-WordNet
UBY
simplified Lesk
degree centrality
results and statistics
sense inventory
corpus reader
answer key annotator
linguistic annotator
WSD annotator
WSD annotator
evaluator

To summarize…

TECHNISCHE UNIVERSITÄT DARMSTADT

DKPro
A comprehensive ecosystem to draw from

… the underlying question ...

Interoperability
Automatic processing
Known tasks

Flexibility
Manual annotation
Novel tasks

Where is the sweet spot?

DKPro Core
UBY
…

WebAnno
DKPro TC
…

UIMA

What is UIMA?

TECHNISCHE UNIVERSITÄT DARMSTADT

- UIMA = Unstructured Information Management Architecture
- A component-based architecture for analysis of unstructured information (e.g., natural language text)
- “Analysis” means deriving a structure from the unstructured data

What is UIMA?

- UIMA = Unstructured Information Management Architecture
- A component-based architecture for analysis of unstructured information (e.g., natural language text)
- “Analysis” means deriving a structure from the unstructured data
- Works like an assembly line:
  - Take the raw material
  - Assemble it step by step
  - Drive off with a nice car

What is UIMA?

TECHNISCHE UNIVERSITÄT DARMSTADT

Accelerating Corporate Research
in the Development, Application and Deployment
of Human Language Technologies

David Ferucci & Adam Lally

Proc. Workshop on Software Engineering and Architecture of Language Technology Systems, 2003

- **Data model** for managing and exchanging unstructured data and annotations
- **Component model** for flexible analytics
- **Process model** for deploying and running analytics
- **Metadata** model to describe all the above
- **Tooling** to run and scale out analytics and to inspect results

https://uima.apache.org

Apache UIMA
History

- 2003 – Ferrucci & Lally paper
- 2004 – IBM alphaWorks project
  - still used e.g. in IBM LanguageWare
- 2006 – Apache Incubator project
- 2009 – OASIS Standard
- 2010 – Full Apache project
- 2010 – Used in IBM's *Watson* Jeopardy Challenge

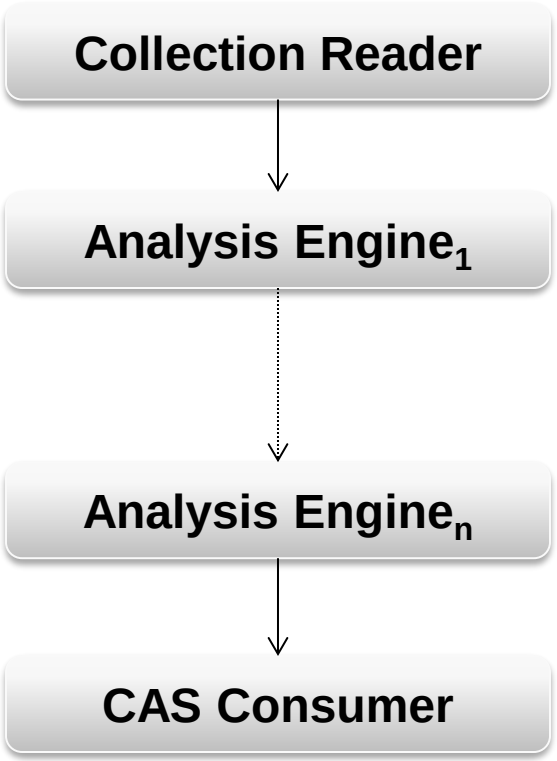
- Various UIMA workshops at COLING, LREC, GSCL, …

- Current version: 2.9.0
- Slowly preparing for version 3...

Apache UIMA

UIMA Aggregate Analysis Engine

- An aggregation of UIMA components
- Specifies a “source to sink” flow of data:

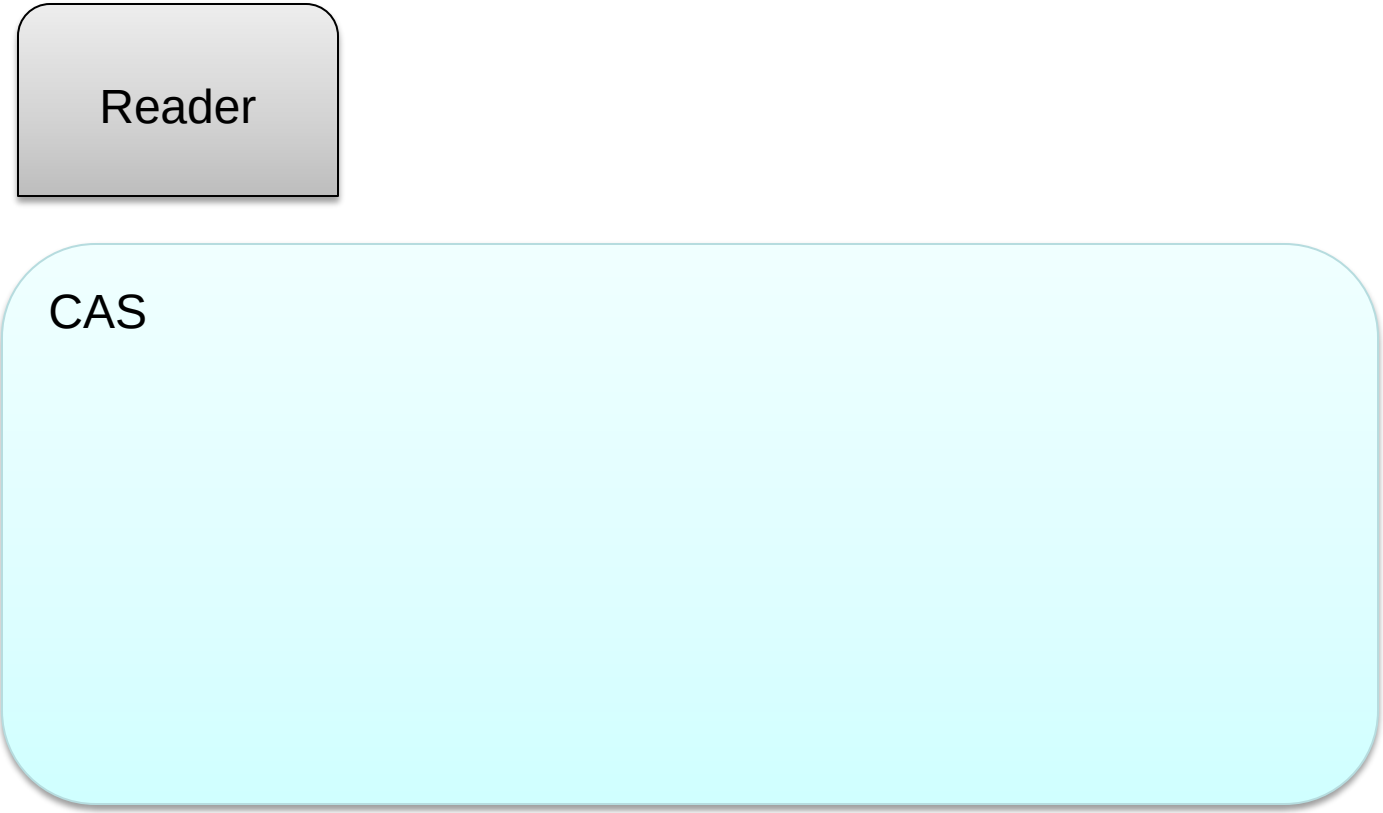


Component – Collection Reader

- Iterates through a source collection to acquire documents

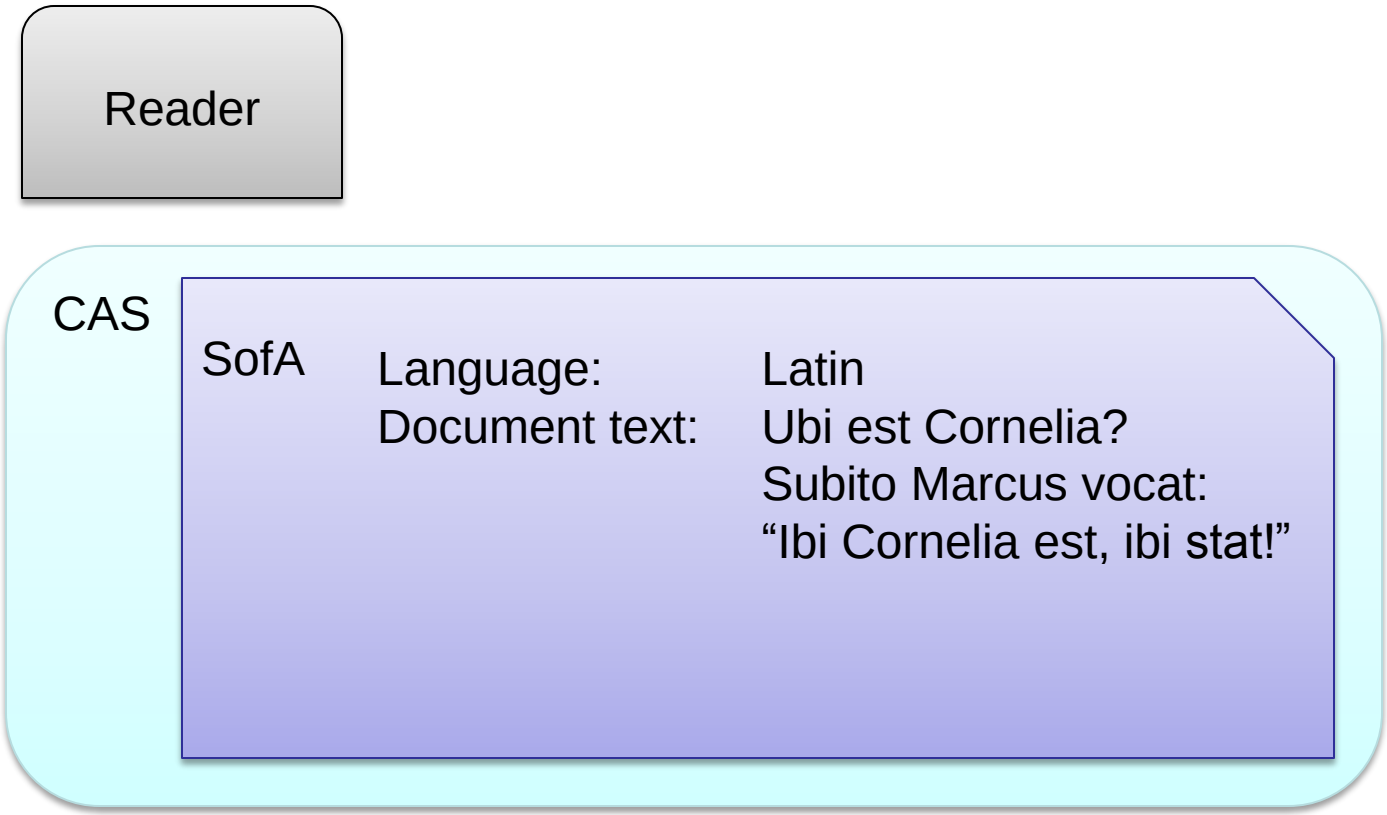
Component – Collection Reader

- Initializes **Common Analysis Structures (CAS)**, generic data structures that hold objects, values, and properties



Component – Collection Reader

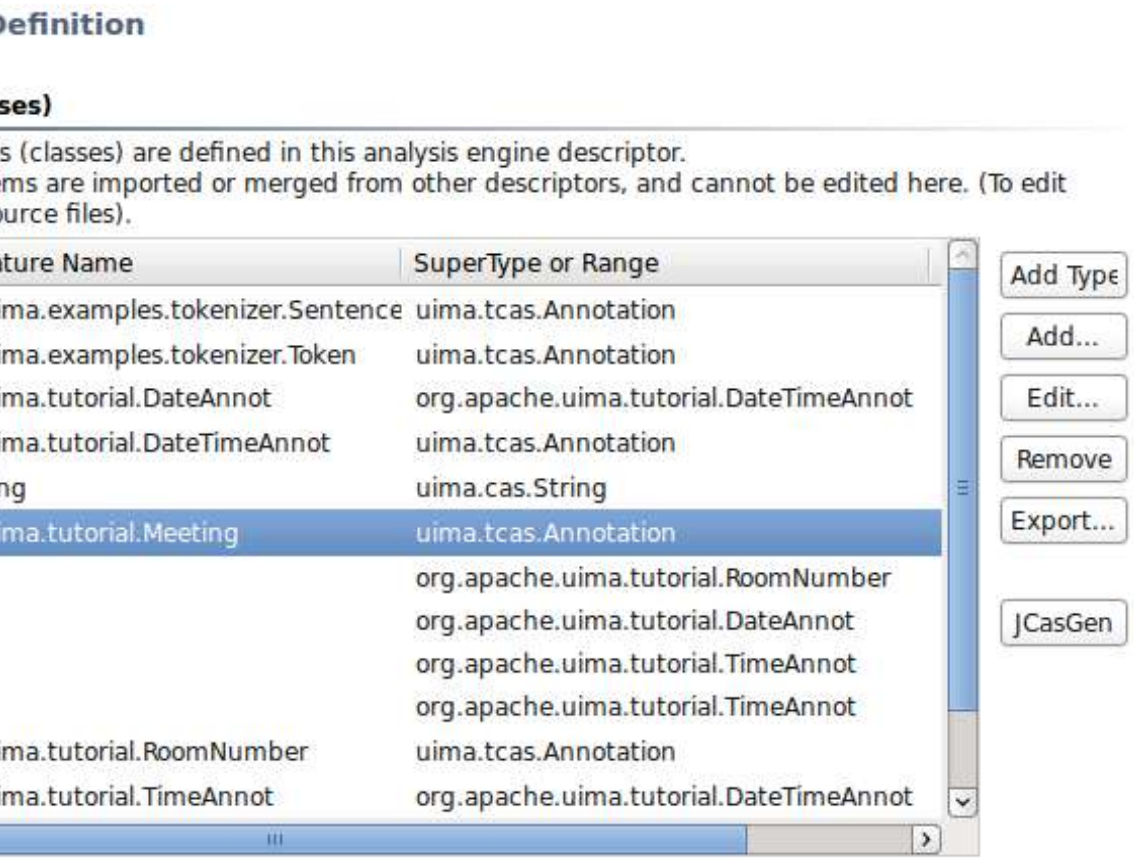
- Each CAS has one or more **views**, each corresponding to a **Subject of Analysis (SofA)**



Types

- UIMA defines a few basic **types**
- Types have properties or **features**
  - Example: We could define a type “Person” which has features such as “Age” and “Gender”
- Types can be extended to define arbitrarily rich domain- and application-specific **type systems**
- A type system defines the various kinds of objects that may be discovered by components that subscribe to that type system
- The (frequently subclassed) **Annotation** type is used to label regions of a document
- Annotations include “begin” and “end” features

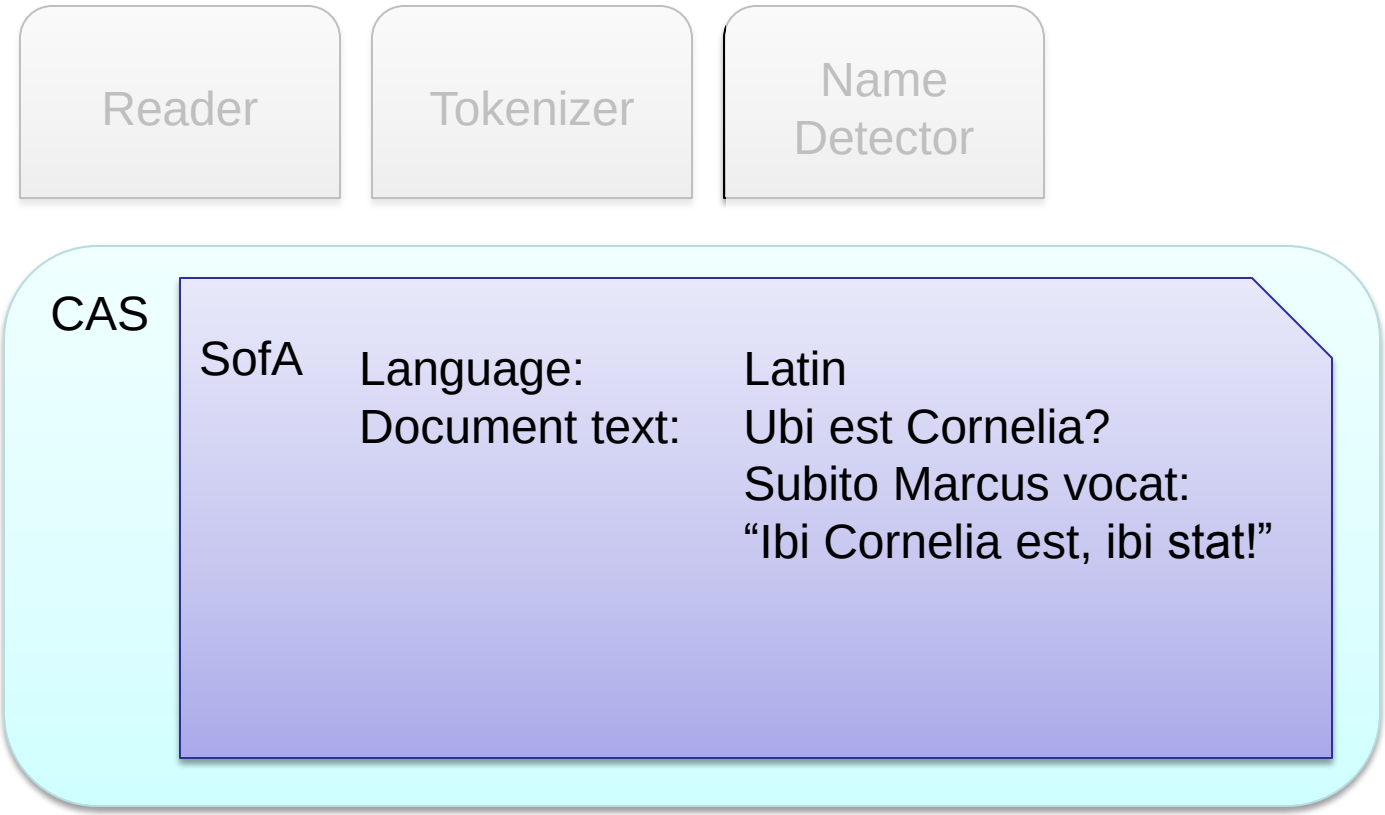
Apache UIMA
Types



Apache UIMA

Component – Analysis Engine

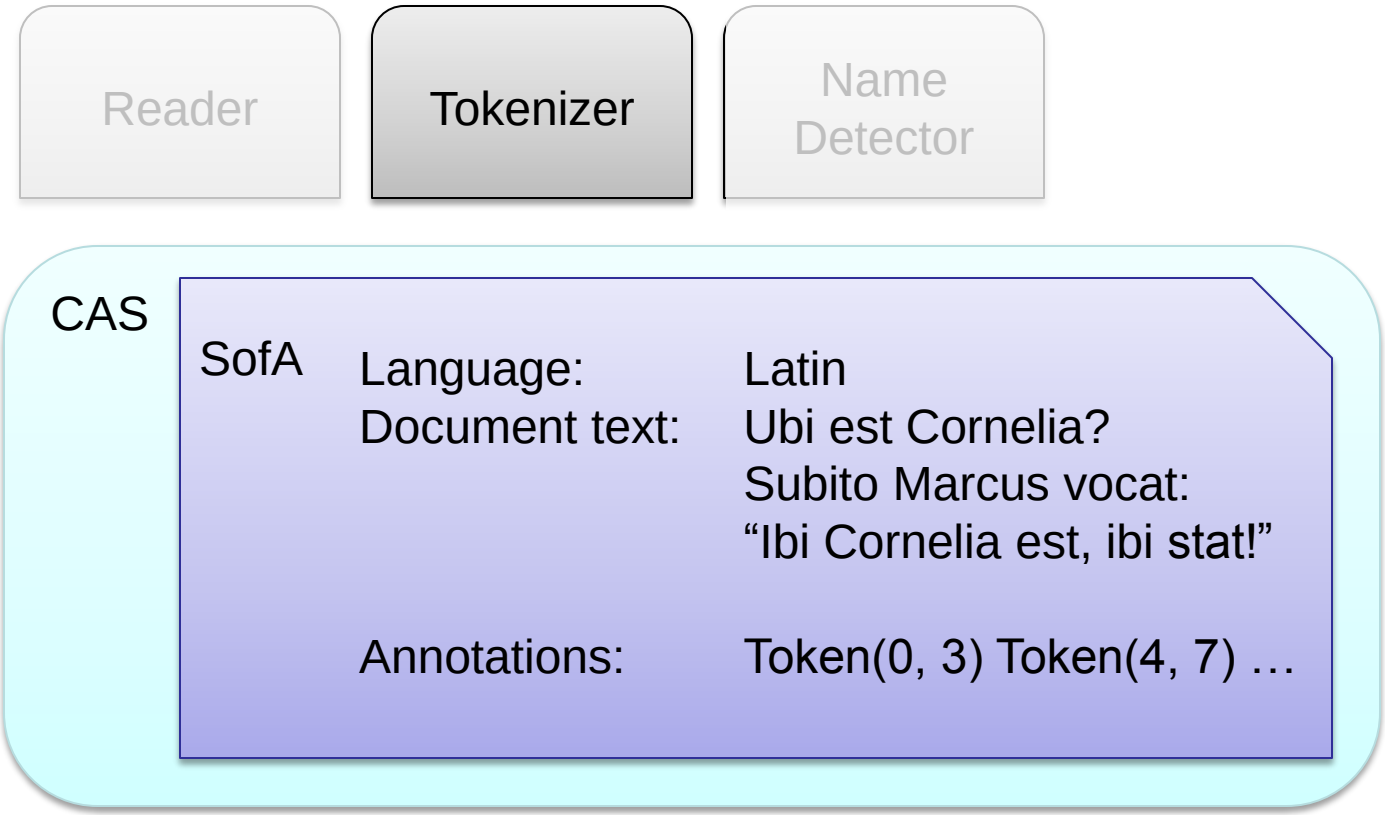
- The structure is passed to one **Analysis Engine (AE)** after the other
- Each AE derives a bit of structure and records it as an Annotation



Apache UIMA

Component – Analysis Engine

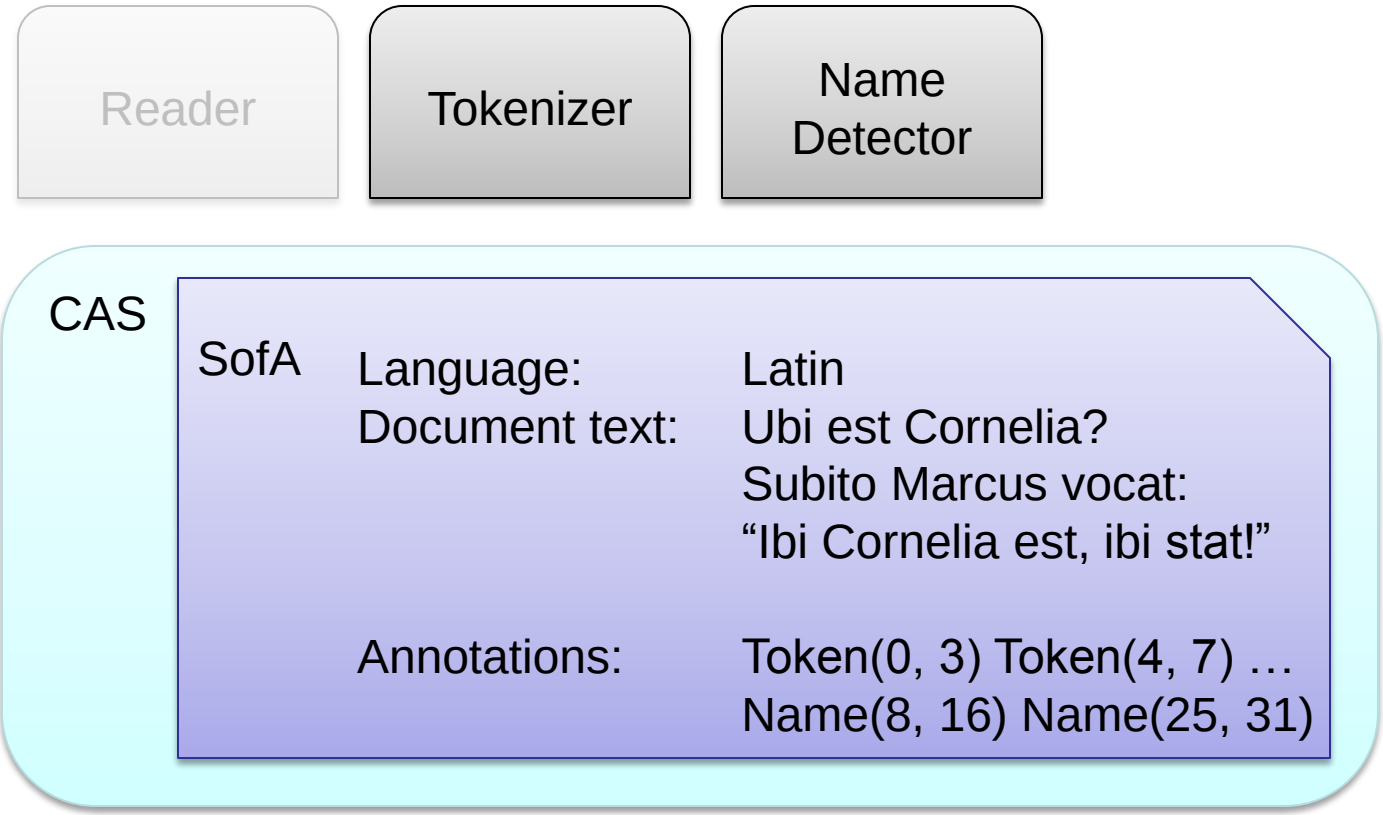
- The structure is passed to one **Analysis Engine (AE)** after the other
- Each AE derives a bit of structure and records it as an Annotation



Apache UIMA

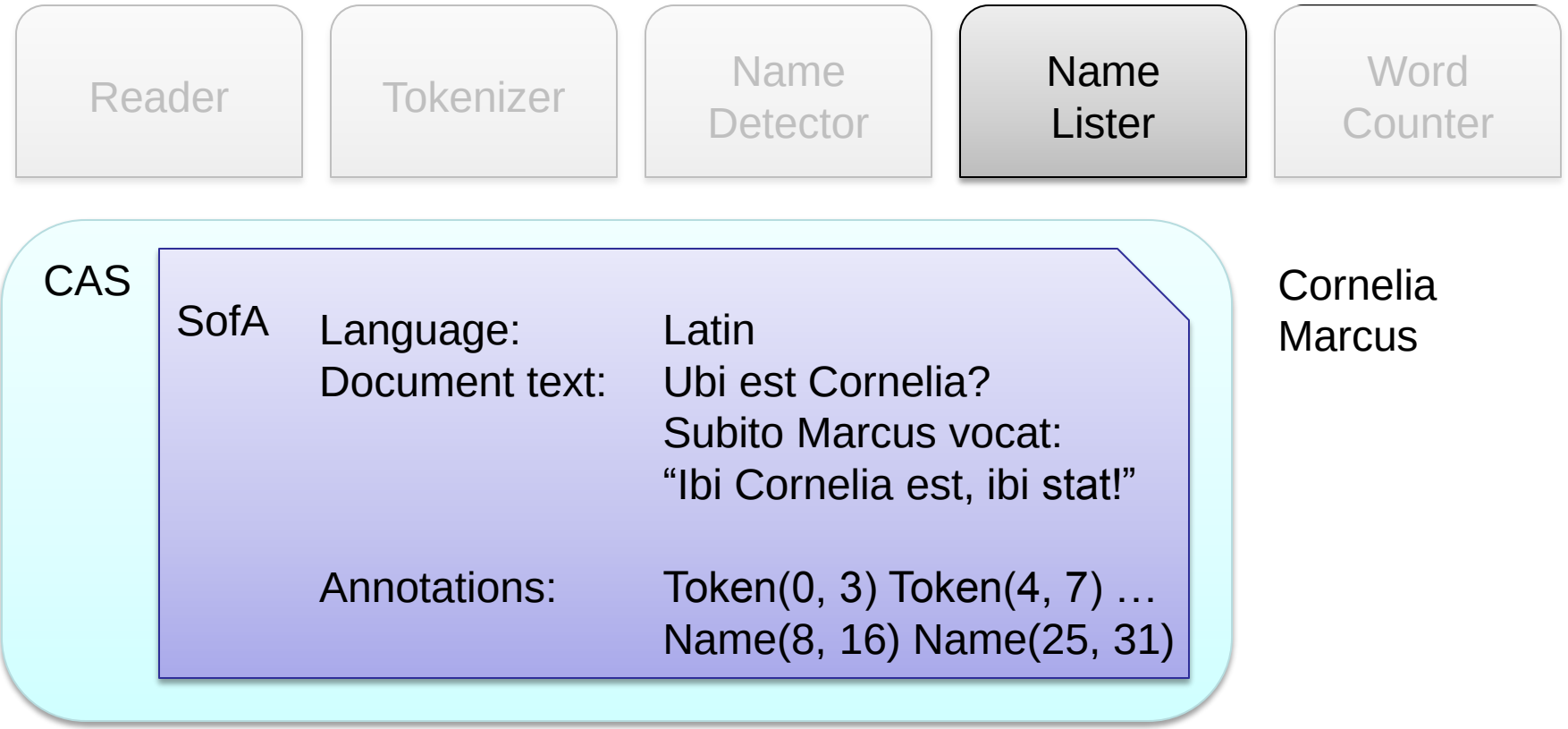
Component – Analysis Engine

- The structure is passed to one **Analysis Engine (AE)** after the other
- Each AE derives a bit of structure and records it as an Annotation



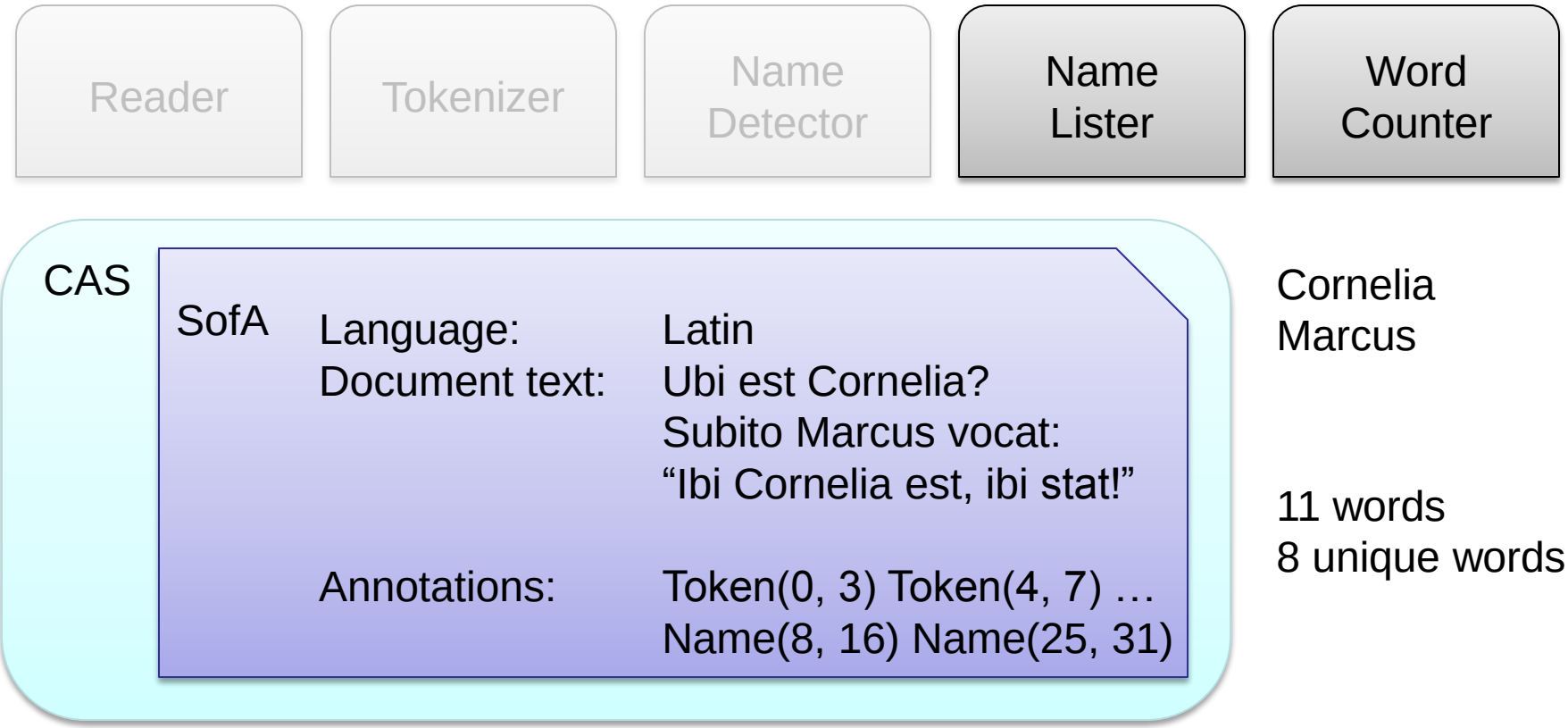
Component – CAS Consumer

- **CAS Consumers** do the final CAS processing
- They can extract, analyze, display, and/or store annotations of interest

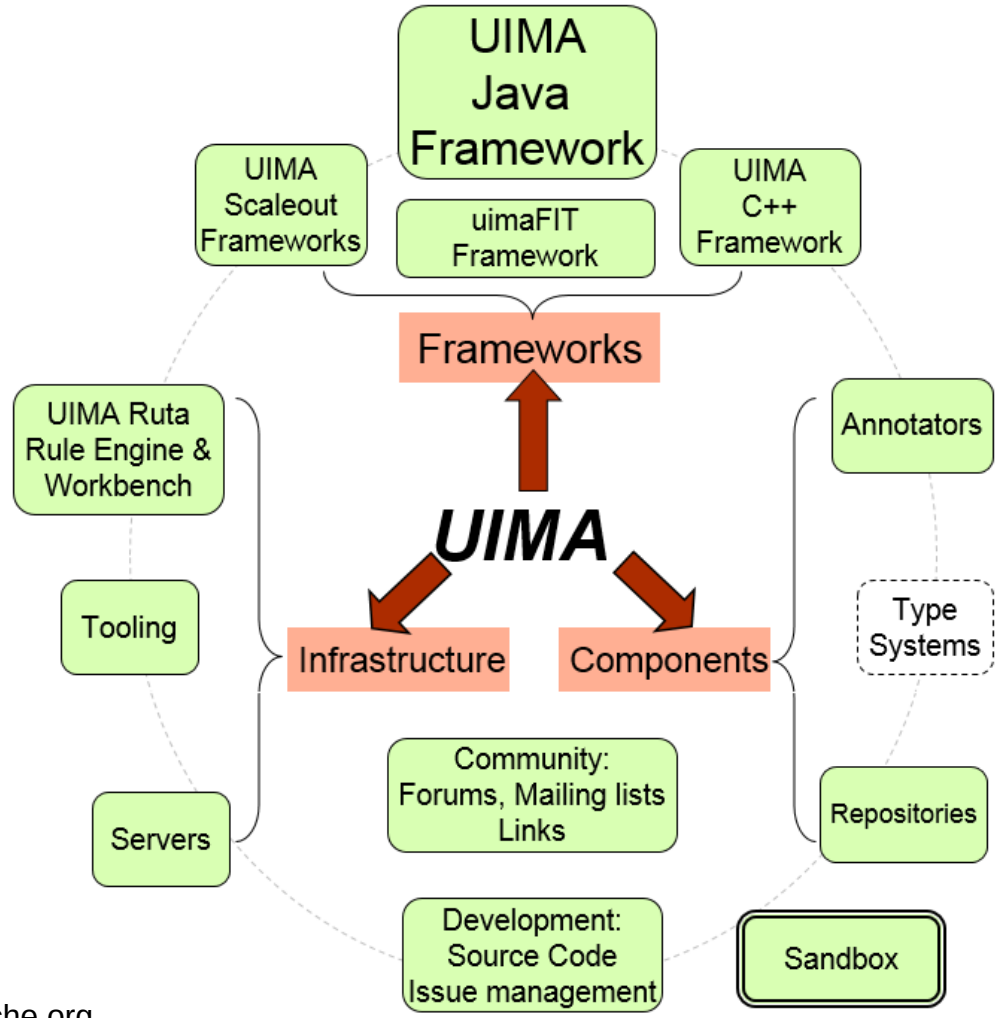


Component – CAS Consumer

- **CAS Consumers** do the final CAS processing
- They can extract, analyze, display, and/or store annotations of interest



Apache UIMA
Was that all?



Source: https://uima.apache.org

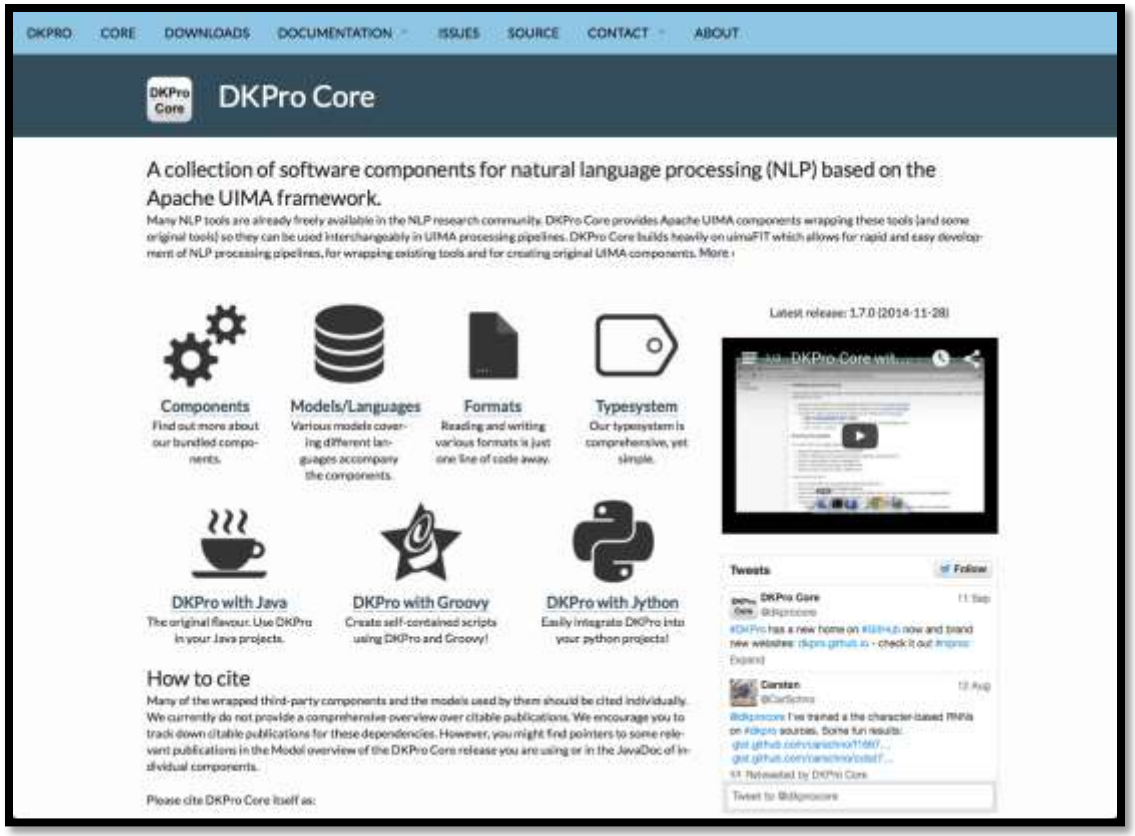
DKPRO CORE

DKPro Core

UIMA-based linguistic preprocessing

TECHNISCHE UNIVERSITÄT DARMSTADT

- NLP
- Normalization
- Preprocessing for ML
- Mix & match components
- Convert between formats



- Train models (new)
- Evaluate (new)

- Experimental pipelines
- Embed in applications
- Ready to run on server/cluster

https://dkpro.github.io/dkpro-core

DKPro Core
History

TECHNISCHE UNIVERSITÄT DARMSTADT

- 2007 project founded
- 2009 first closed-source release of DKPro Core (1.0)
- 2011 the first open-source release of DKPro Core (1.1.0) published on Google Code
- 2012 first published via Maven Central
- 2014 becoming a community project
  adopted contributor licence agreement
  started accepting external contributions
- 2015 migration to GitHub

- Latest release 1.8.0 (22 June 2016)
- Upcoming release 1.9.0 (probably this year)

Building blocks (1.8.0 → 1.9.0)

Formats
(49 → 59)

Components
(94 → 138)

Models
(218 → 267)

Type System

Tagsets
(66 → 77)

Datasets (0 → 42)

New in
1.9.0

DKPro Core
Readers and writers

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Common parameters**
  - Source / target location
  - Source / target encoding
  - Ant-like patterns (for readers)
  - Language (for readers)
  - Tagset mapping
  - Control reading/writing of individual layers
  - …

- **Common features**
  - Read data from file system, ZIP/JAR archives or classpath
  - Support for other file systems pluggable (e.g., HDFS)
  - Preserve directory structure on write for recursive reads

DKPro Core
Components

- Checker (spelling/grammar)
- Chunker
- Coreference resolver
- Embeddings
- Gazeteer
- Language identifier
- Lemmatizer
- Morphological analyzer
- Named entity recognizer
- Parser
- Part-of-speech tagger
- Phonetic transcriptor
- Segmenter
- Semantic role labeller
- Stemmer
- Topic model
- Transformer/normalization
- ...

- Suites
  - Apache OpenNLP
  - ClearNLP
  - Emory NLP4J
  - Stanford CoreNLP
  - Illinois CogComp NLP
  - Mate Tools
  - LanguageTool
  - …
- Standalone tools
  - Malt Parser
  - Mst Parser
  - Berkeley Parser
  - TreeTagger
  - RfTagger
  - SFST
  - …

DKPro Core
Example Pipeline

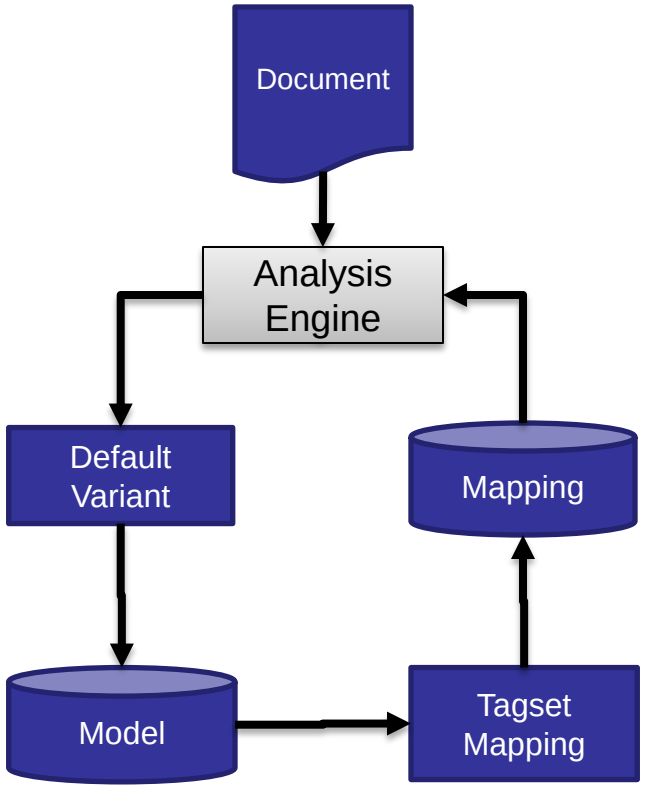
TECHNISCHE UNIVERSITÄT DARMSTADT

```
SimplePipeline.runPipeline(
  createReaderDescription(TextReader.class,
    TextReader.PARAM_SOURCE_LOCATION, “texts/**/*.txt”
    TextReader.PARAM_LANGUAGE, “en”),

  createEngineDescription(OpenNlpSegmenter.class),
  createEngineDescription(MatePosTagger.class),
  createEngineDescription(ClearNlpLemmatizer.class),
  createEngineDescription(BerkeleyParser.class,
    BerkeleyParser.PARAM_WRITE_PENN_TREE, true),
  createEngineDescription(StanfordNamedEntityRecognizer.class),

  createEngineDescription(XmiWriter.class,
    XmiWriter.PARAM_TARGET_LOCATION, “output”,
    XmiWriter.PARAM_TYPE_SYSTEM_FILE, “TypeSystem.xml”);
```

Model loading

- **Common parameters**
  - Model location
  - Model encoding
  - Model variant
  - Mapping location
  - Language

- **Common features**
  - Load model depending on document language
  - Print model tag set to log
  - Default variants
  - Download model automatically (optional)



classpath:/de/tudarmstadt/ukp/dkpro/core/opennlp/lib/tagger-**${language}**-**${variant}**.bin
classpath:/de/tudarmstadt/ukp/dkpro/core/api/lexmorph/tagset/**${language}**-**${pos.tagset}**-pos.map

DKPro Core
Normalization

TECHNISCHE UNIVERSITÄT DARMSTADT

- Changes to text are not allowed in UIMA
- Normalization usually happens inside the components
  - Different components may require different normalizations
- SurfaceForm – annotate normalized text with original text
  - Used in CoNLL-U reader/writer and WebAnno
- DKPro Core Text Normalizer components
  - Creates a new, modified document (or a new view in the same document)
  - Hyphenation removal, PTB normalization, spelling correction, …

DKPro Core
Datasets (1.9.0+)

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Common features**
  - Downloading and caching
  - Pre-defined train/development/test data
  - Generation of splits
  - Extraction of archives
- Growing number of dataset descriptions come with DKPro Core
- … or define your own within your experiment / project

DKPro Core
Datasets (1.9.0+)

TECHNISCHE UNIVERSITÄT DARMSTADT

Georgetown University Multilayer Corpus

Edit on GitHub

ID	gum-en-conll-2.2.0
Version	2.2.0
Media type	text/x.org.dkpro.conll-2006
Language	en
Encoding	UTF-8
URL	https://corpling.uis.georgetown.edu/gum/
Attribution[1]	Zeldes, Amir (2016) "The GUM Corpus: Creating Multilayer Resources in the Classroom". Language Resources and Evaluation. For Gum annotation team, see https://corpling.uis.georgetown.edu/gum/
License[2]	*CC-BY 2.5*, *CC-BY-SA 3.0*, *CC-BY-NC-SA 3.0*, *CC-BY 4.0*

Table 17. License comments for Georgetown University Multilayer Corpus

License	Comment
CC-BY 2.5	Wikinews texts (Source: https://en.wikinews.org/wiki/Wikinews:Copyright)
CC-BY-SA 3.0	WikiVoyage texts (Source: https://wikimediafoundation.org/wiki/Terms_of_Use)
CC-BY-NC-SA 3.0	WikiVoyage texts (Source: http://www.wikihow.com/wikiHow:Creative-Commons)
CC-BY 4.0	Annotations (Source: https://corpling.uis.georgetown.edu/gum/)

Table 18. Artifacts for Georgetown University Multilayer Corpus

Artifact	SHA1
gum.zip	b17e276998ced83153be605d8157afacf1f10fdc

DKPro Core

Training models (1.9.0+)

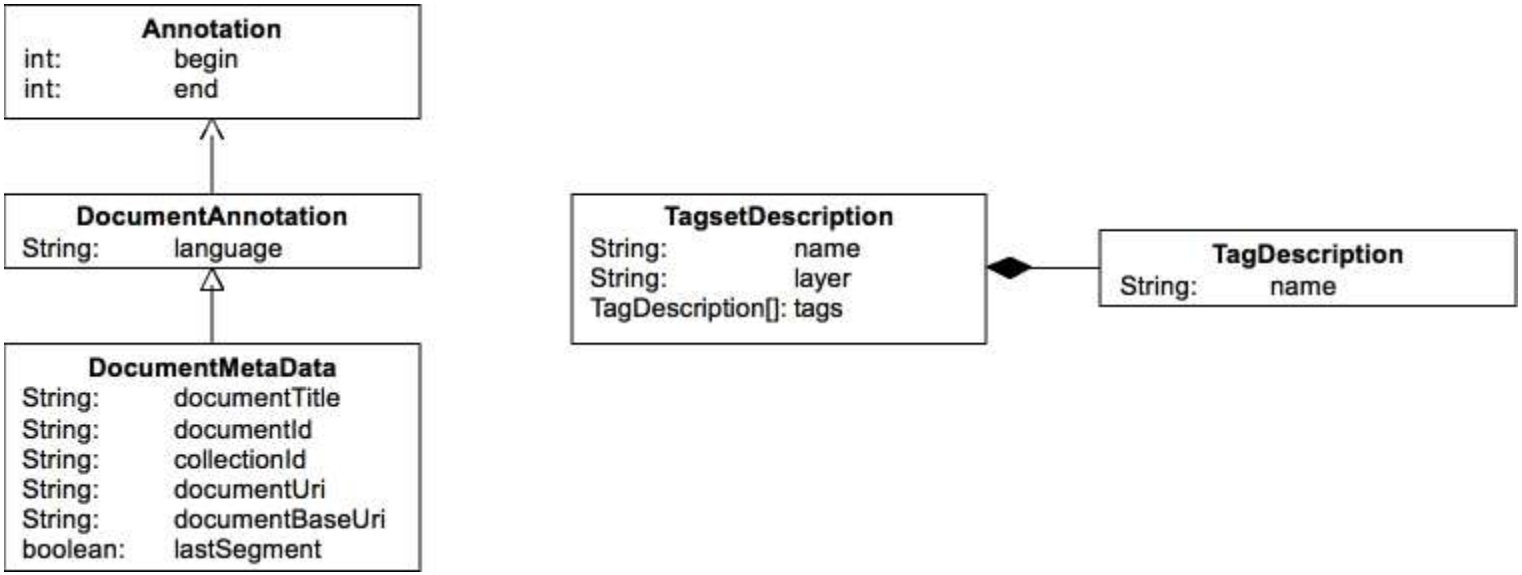
TECHNISCHE UNIVERSITÄT DARMSTADT

- Starting to include training components
  - OpenNLP (segmenter, POS tagger, chunker, NER)
  - Stanford CoreNLP (POS tagger)
  - … more to come
- Basic evaluation framework included

```
DatasetFactory loader = new DatasetFactory(cache);
Dataset ds = loader.load("gum-en-conll-2.2.0");
Split split = ds.getSplit(0.8);

CollectionReaderDescription trainReader = createReaderDescription(
        Conll2006Reader.class,
        Conll2006Reader.PARAM_PATTERNS, split.getTrainingFiles(),
        Conll2006Reader.PARAM_USE_CPOS_AS_POS, true,
        Conll2006Reader.PARAM_LANGUAGE, ds.getLanguage());

AnalysisEngineDescription trainer = createEngineDescription(
        OpenNlpPosTaggerTrainer.class,
        OpenNlpPosTaggerTrainer.PARAM_TARGET_LOCATION, new File(targetFolder, "model.bin"),
        OpenNlpPosTaggerTrainer.PARAM_LANGUAGE, ds.getLanguage());

SimplePipeline.runPipeline(trainReader, trainer);
```

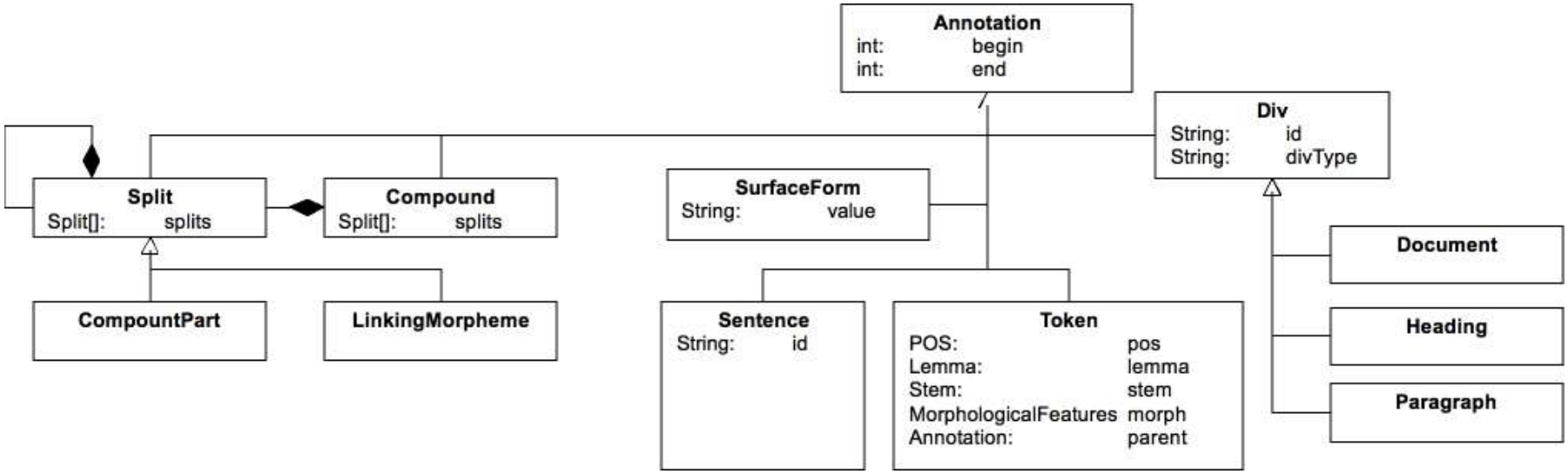
TYPE SYSTEM

DKPro Core Type System
Metadata



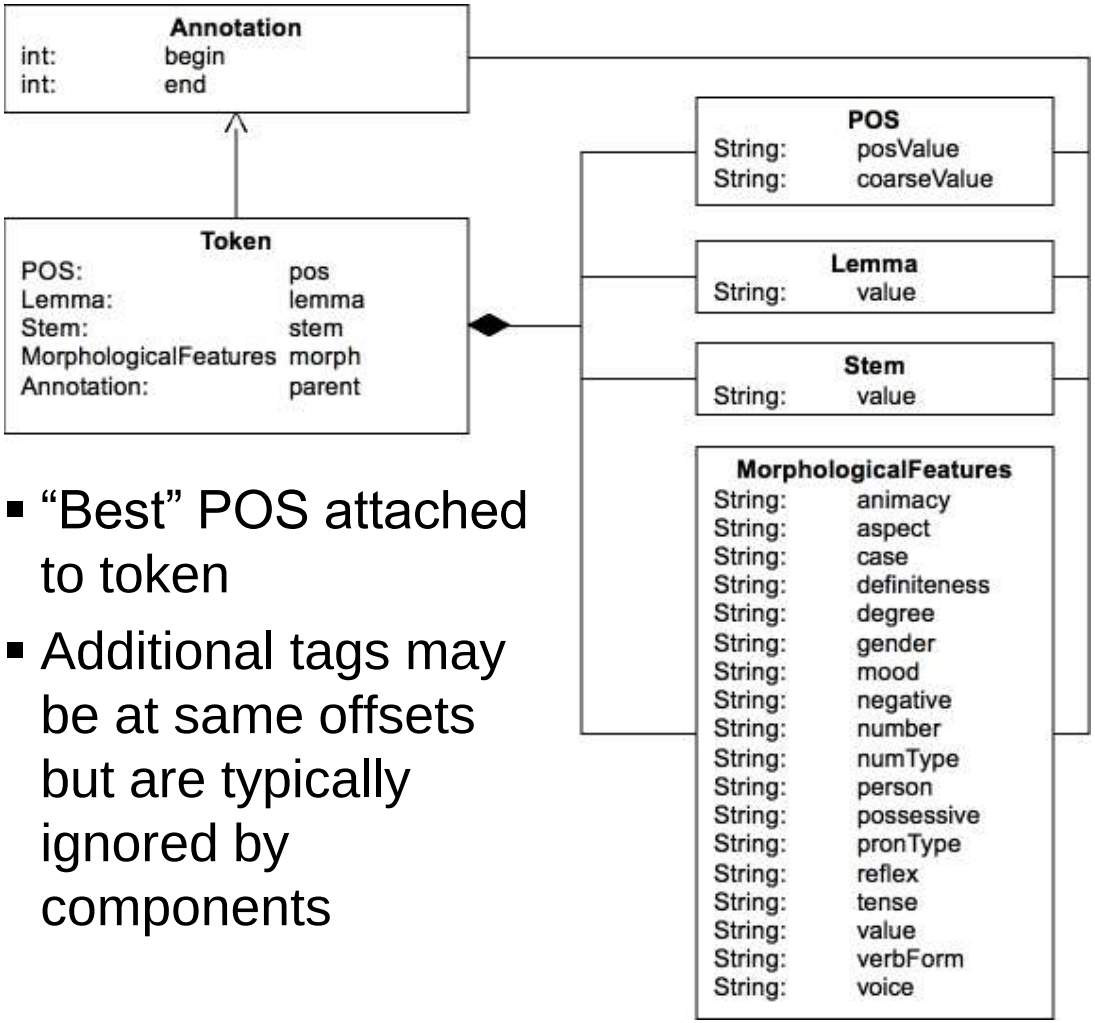
- **DocumentMetaData** created by **readers**, essential for **writers**
  - Reconstruction of recursive folder structures
- **TagsetDescription** / **TagDescription** extracted from **models**

DKPro Core Type System
Segmentation



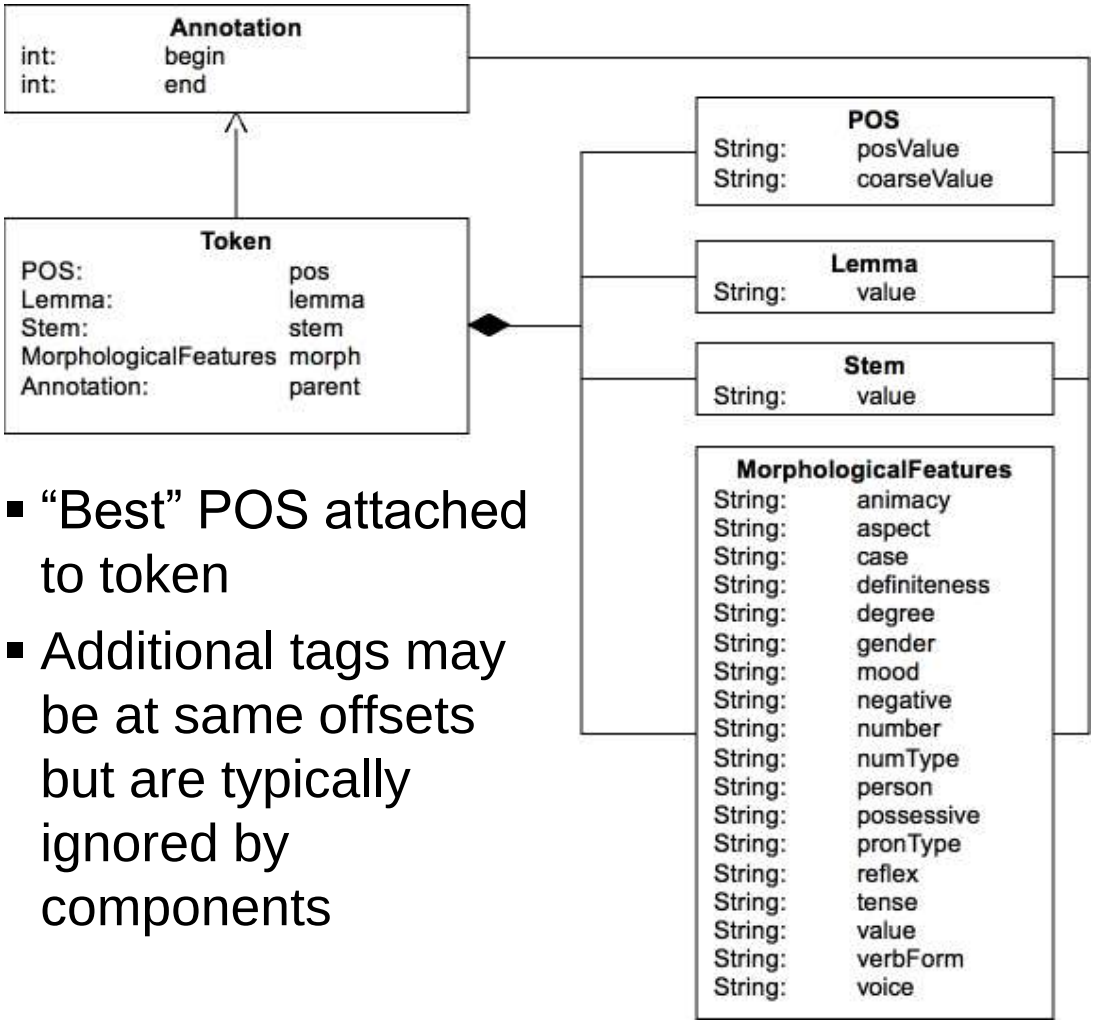
- Each document has one set of segmentation annotations
- **id** externally assigned – just passed through

DKPro Core Type System
Token and attached information

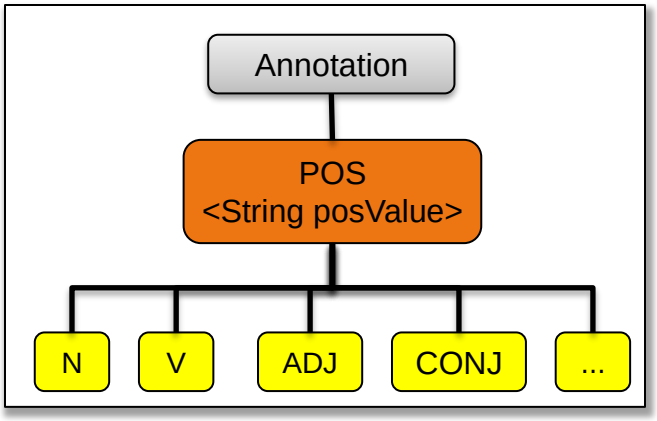


- "Best" POS attached to token
- Additional tags may be at same offsets but are typically ignored by components

DKPro Core Type System
Token and attached information

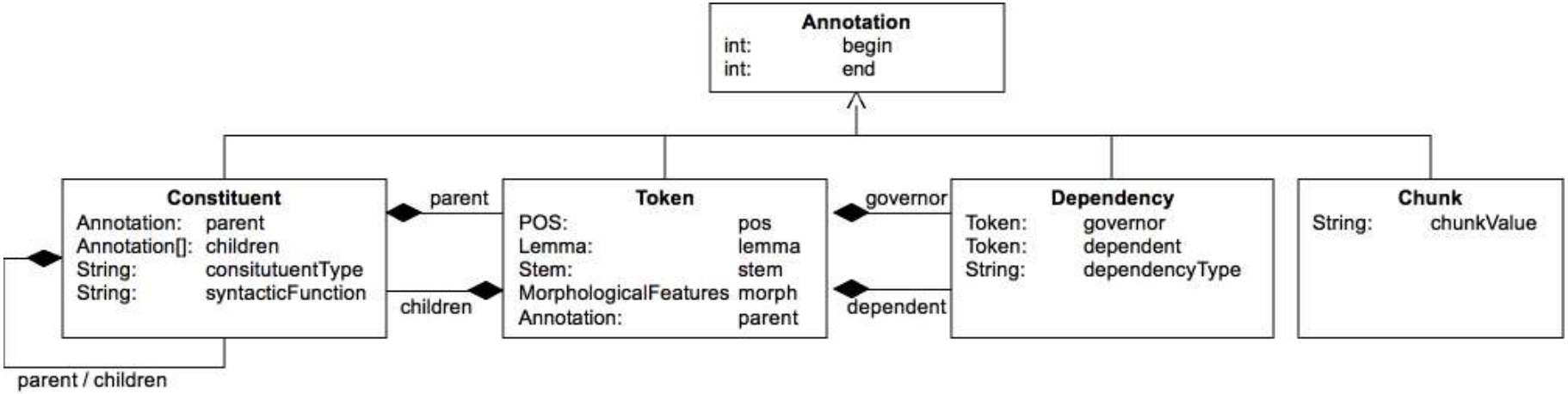


- "Best" POS attached to token
- Additional tags may be at same offsets but are typically ignored by components



- Using "elevated types"
- UD POS tags
- Similar for
  - Dependencies
  - Constituents
  - Named entities

DKPro Core Type System
Syntax

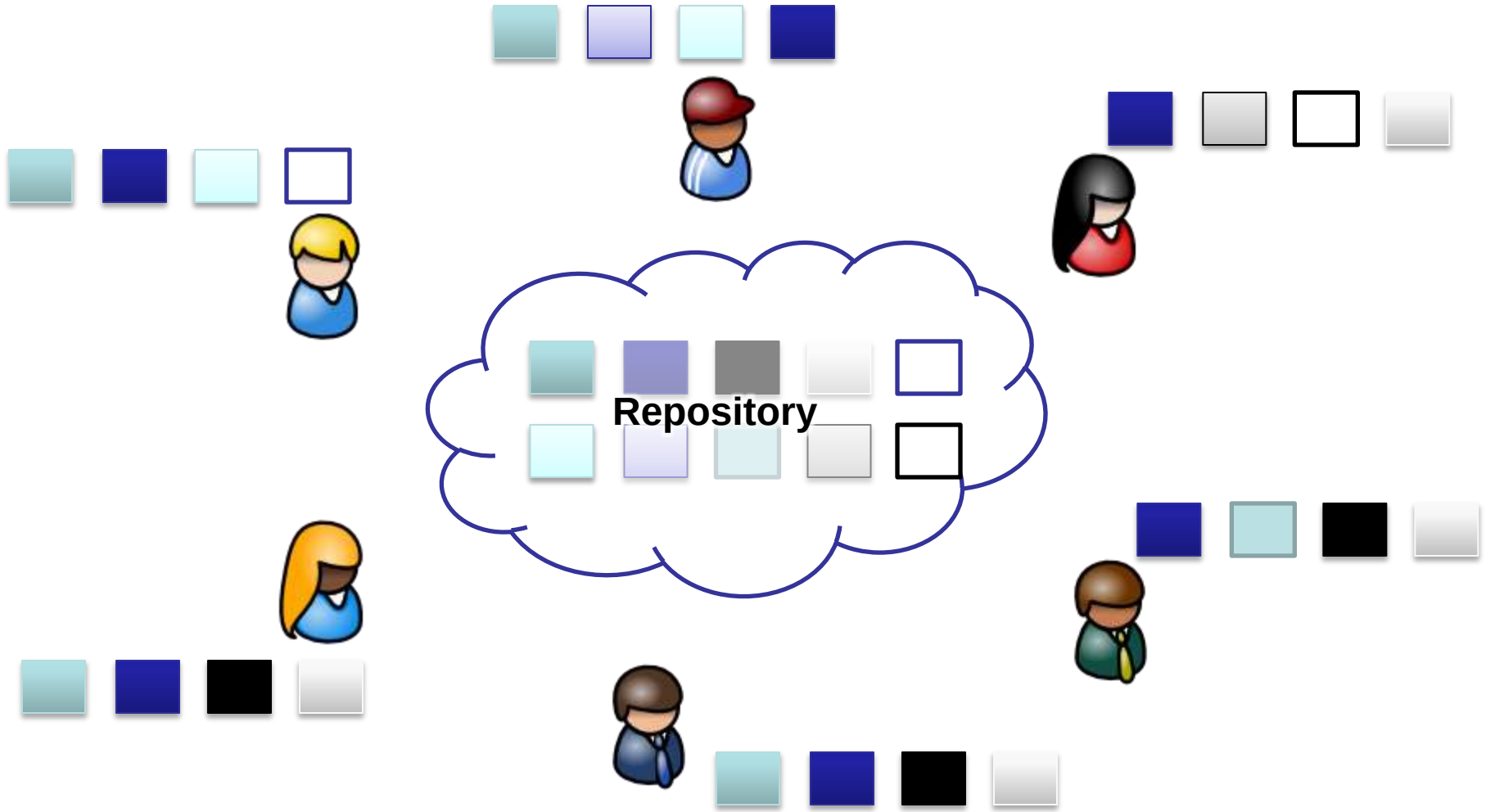


- Conventions
  - Constituent: parent/child features consistent
  - Constituent: root constituent has type **ROOT**
  - Dependencies: root dependency has type **ROOT** and is its own governor

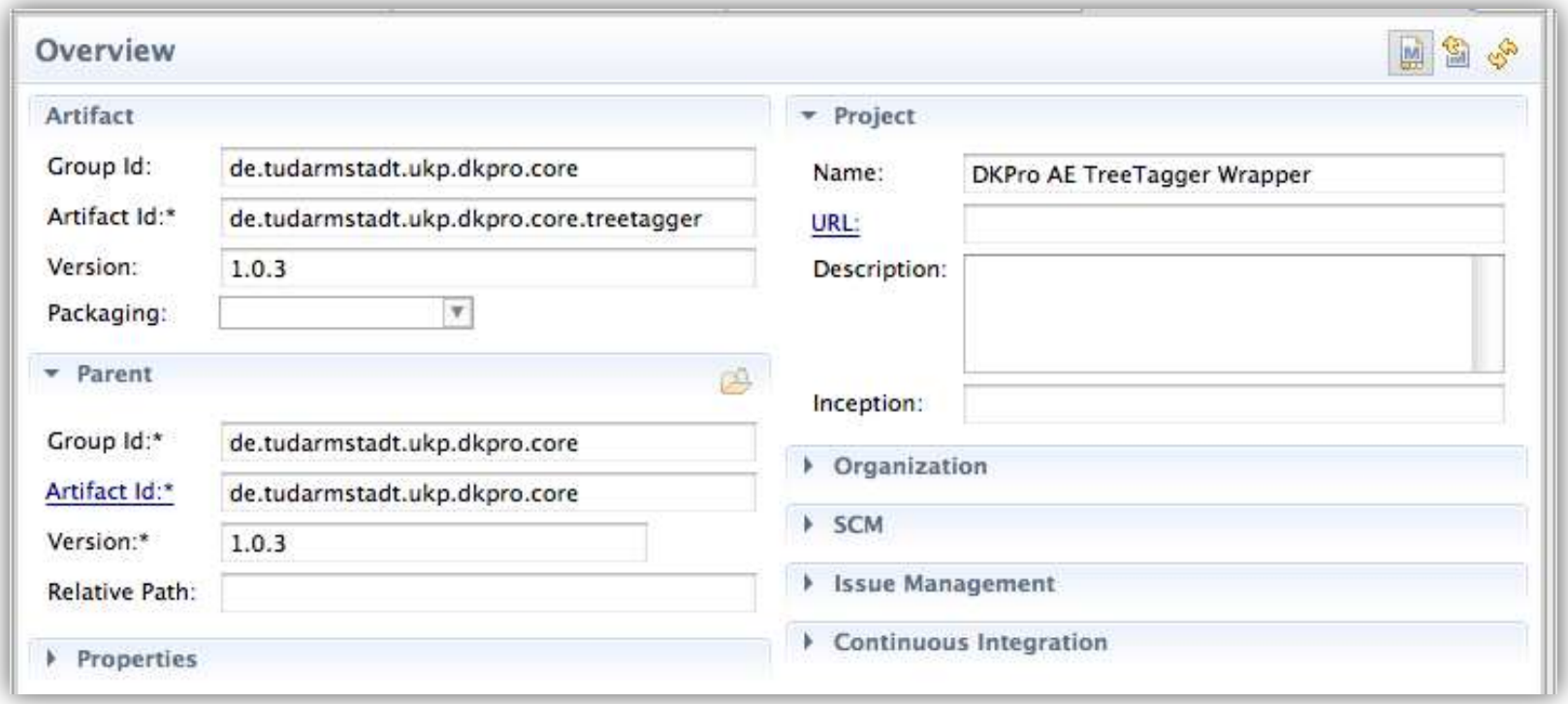
TECHNISCHE UNIVERSITÄT DARMSTADT

THE LONG WINDING ROAD TOWARDS USABILITY…

UKP Software Repository



Publishing reusable components



Automatic quality testing

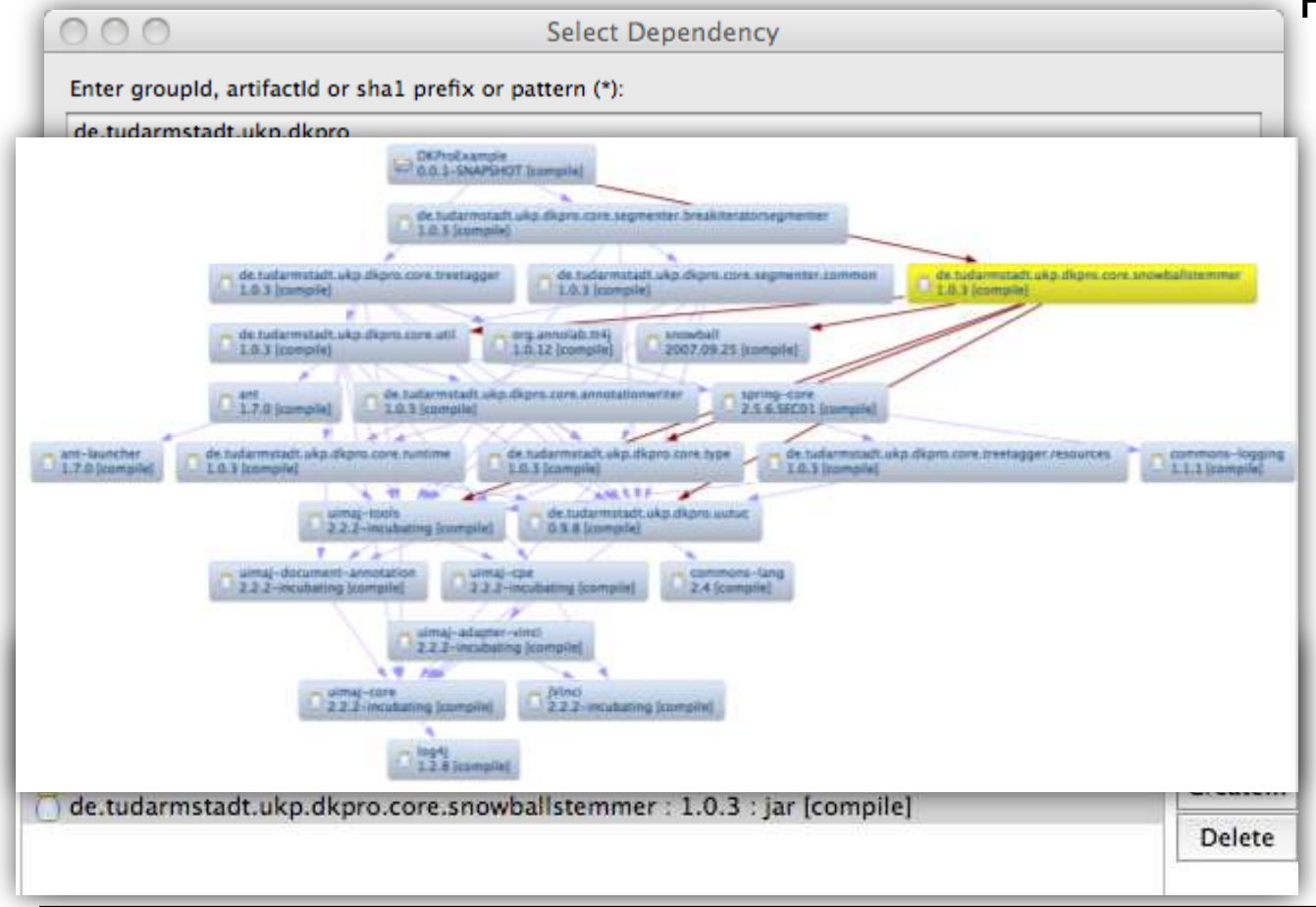
Source Version Control System

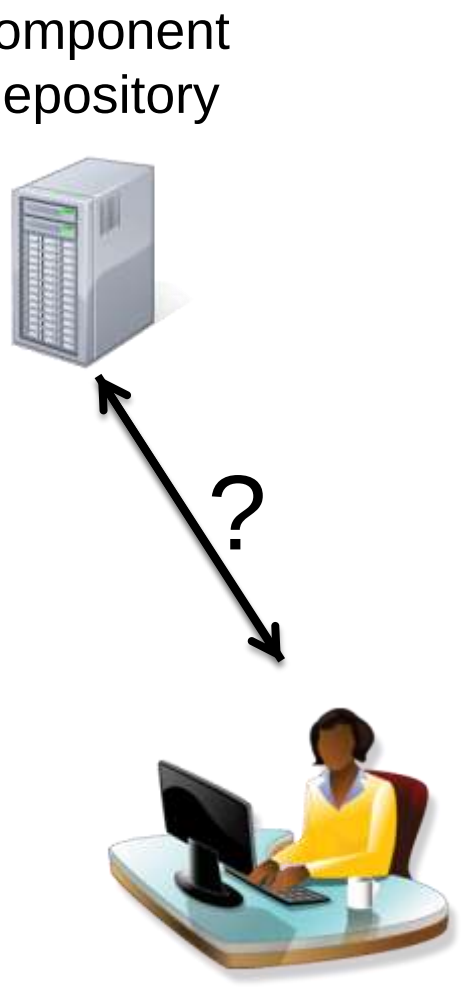
Automatic Building & Testing

Component Repository

- Current development snapshots
- Stable release versions
- Searchable via web interface
- Seamless integration with development environment

Using the components





Overview

Development environment	Eclipse
Project management	Maven / m2eclipse
Source version control	Git / GitHub / Egit / Sourcetree
Building and testing	Jenkins
Artifact repository	Artifactory
Issue tracking	GitHub
Mailing lists	Google Groups

Problems

TECHNISCHE UNIVERSITÄT DARMSTADT

- … sounds very good but …

- **UIMA difficult to develop**
  - Verbose code
  - Extensive use of XML descriptors
  - Java code and descriptors get out of sync

- **UIMA difficult to use**
  - Tools often based on XML descriptors
  - Graphical tools do not connect to component repository
  - Eclipse / Maven not convenient

How to avoid inheriting these problems in DKPro Core?

Apache uimaFIT

TECHNISCHE UNIVERSITÄT DARMSTADT

- Create and configure pipelines easily in Java
- Test UIMA components

- Started out as a collaborative effort between
  - Center for Computational Pharmacology, University of Colorado, Denver
  - Center for Computational Language and Education Research, University of Colorado, Boulder,
  - Ubiquitous Knowledge Processing (UKP) Lab, Technische Universität Darmstadt

- Since version 2.0.0 part of the Apache UIMA project

https://uima.apache.org/uimafit.html

Important features of uimaFIT

TECHNISCHE UNIVERSITÄT DARMSTADT

uimaFIT is key to make UIMA usable within Java code

- **Factories – dynamic assembly of analysis pipelines**
  - Automatic type system detection
  - Most metadata maintained in Java
  - Refactorable code
- **Injection – convenient implementation of analysis components**
  - Default parameter values
  - Parameter types not supported by UIMA (e.g., File, URL, …)
- **Testing – easy running of analysis pipelines**
  - Unit tests easy to set up
  - … or research experiments
- **Building – enhanced UIMA/Java integration**
  - Inject Maven metadata into UIMA metadata (e.g., version, vendor, etc.)
  - Extract Javadocs from sources and inject them into UIMA metadata
  - Generate component descriptors at build time (experimental)

… and more …

Navigating the CAS with JCasUtil/CasUtil

TECHNISCHE UNIVERSITÄT DARMSTADT

- select(cas, type)
- selectAll(cas)
- selectSingle(cas, type)
- selectSingleRelative(cas, type, n)
- selectBetween(type, annotation1, annotation2)
- selectCovered(type, annotation)
- selectCovering(type, annotation)
- selectByIndex(cas, type, n)
- selectPreceeding(type, annotation, n)
- selectFollowing(type, annotation, n)

```
for (Token token : JCasUtil.select(jcas, Token.class)) {
  ...
}
```

Code: process() (uimaFIT)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public static final String PARAM_DICTIONARY_FILE = "dictionaryFile";
@ConfigurationParameter(name = PARAM_DICTIONARY_FILE, mandatory = true)
private File dictionaryFile;

private Set<String> names;

public void initialize(UimaContext aContext)
{
  super.initialize(aContext);
  names = new HashSet<String>(readLines(dictionaryFile));
}

public void process(JCas jcas)
{
  // Annotate tokens contained in the dictionary as name
  for (Token token : select(jcas, Token.class)) {
    if (names.contains(token.getCoveredText())) {
      new Name(jcas, token.getBegin(), token.getEnd()).addToIndexes();
    }
  }
}
```

Code: UIMA JCas

```
TypeSystemDescription tsd = new TypeSystemDescription_impl();
TypeDescription tokenTypeDesc = tsd.addType("Token", "", CAS.TYPE_NAME_ANNOTATION);
tokenTypeDesc.addFeature("length", "", CAS.TYPE_NAME_INTEGER);

JCas jcas = CasCreationUtils.createCas(tsd, null, null).getJCas;
jcas.setDocumentText("This is a test.");


new Token(jcas, 0, 4).addToIndexes();
new Token(jcas, 5, 7).addToIndexes();
new Token(jcas, 8, 9).addToIndexes();
new Token(jcas, 10, 14).addToIndexes();
new Token(jcas, 14, 15).addToIndexes();


AnnotationIndex<AnnotationFS> tokenIdx = cas.getAnnotationIndex(Token.type);
for (AnnotationFS token : tokenIdx) {
  ((Token) token).setLength(token.getCoveredText().length());
}

for (AnnotationFS token : tokenIdx) {
  System.out.println(token.getCoveredText() + " – " + token.getLength);
}
```

Code: uimaFIT JCas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
JCas jcas = JCasFactory.createJCas();
jcas.setDocumentText("This is a test.");


new Token(jcas, 0, 4).addToIndexes();
new Token(jcas, 5, 7).addToIndexes();
new Token(jcas, 8, 9).addToIndexes();
new Token(jcas, 10, 14).addToIndexes();
new Token(jcas, 14, 15).addToIndexes();



for (Token token : select(jcas, Token.class)) {
  token.setLength(token.getCoveredText().length());
}

for (Token token : select(jcas, Token.class)) {
  System.out.println(token.getCoveredText()+" - "+token.getLength());
}
```

Making DKPro Core easy to use

- For hard-core Java developers, Eclipse + Maven is very convenient
- What about others (e.g., Digital Humanities researchers)?
- Requirements
  - Work without Eclipse
  - Work without Maven
  - Simple solutions should fit into a single file

DKPro Core + uimaFIT + Groovy

```
#!/usr/bin/env groovy
@Grab(group='de.tudarmstadt.ukp.dkpro.core',
      module='de.tudarmstadt.ukp.dkpro.core.opennlp-asl',
      version='1.5.0')

import de.tudarmstadt.ukp.dkpro.core.opennlp.*;
import org.apache.uima.fit.factory.JCasFactory;
import org.apache.uima.fit.pipeline.SimplePipeline;
import de.tudarmstadt.ukp.dkpro.core.api.segmentation.type.*;
import de.tudarmstadt.ukp.dkpro.core.api.syntax.type.*;
import static org.apache.uima.fit.util.JCasUtil.*;
import static org.apache.uima.fit.factory.AnalysisEngineFactory.*;

def jcas = JCasFactory.createJCas();
jcas.documentText = "This is a test";
jcas.documentLanguage = "en";

SimplePipeline.runPipeline(jcas,
  createEngineDescription(OpenNlpSegmenter),
  createEngineDescription(OpenNlpPosTagger),
  createEngineDescription(OpenNlpParser,
    OpenNlpParser.PARAM_WRITE_PENN_TREE, true));

select(jcas, Token).each { println "${it.coveredText} ${it.pos.posValue}" }
select(jcas, PennTree).each { println it.pennTree }
```

Fetches all required dependencies

No manual installation!

Input

Analytics pipeline. Language-specific resources fetched automatically

Output

DKPro Core + uimaFIT + Groovy

TECHNISCHE UNIVERSITÄT DARMSTADT

```
#!/usr/bin/env groovy
@Grab(group='de.tudarmstadt.ukp.dkpro.core',
      module='de.tudarmstadt.ukp.dkpro.core.opennlp-asl',
      version='1.5.0')

import de.tudarmstadt.ukp.dkpro.core.opennlp.*;
import org.apache.uima.fit.fa
import org.apache.uima.fit.pipeline.SimplePipeline;
import de.tudarmstadt.ukp.dk
import de.tudarmstadt.ukp.dk
import static org.apache.uim
import static org.apache.uim

def jcas = JCasFactory.creat
jcas.documentText = "This is a test";
jcas.documentLanguage = "e

SimplePipeline.runPipeline(j
  createEngineDescription(OpenNlpSegmenter),
  createEngineDescription(O
  createEngineDescription(OpenNlpParser,
    OpenNlpParser.PARAM_

select(jcas, Token).each { println "${it.coveredText} ${it.pos.posValue}" }
select(jcas, PennTree).each { println it.pennTree }
```

Fetches all required dependencies

No manual installation!

Why is this cool?

This is an actual running example!

Requires only
JVM + Groovy (+ Internet connection)

Input

Trivial to embed in applications
Trivial to wrap as a service

Analytics pipeline. Language-specific resources fetched automatically

Easy to parallelize / scale

Output

Similar solution available for Jython!

Still too complicated?

DKPro Core + uimaFIT + Groovy

```
#!/usr/bin/env groovy
@Grab(group='de.tudarmstadt.ukp.dkpro.core',
      module='de.tudarmstadt.ukp.dkpro.core.opennlp-asl',
      version='1.5.0')

import de.tudarmstadt.ukp.dkpro.core.opennlp.*;
import org.apache.uima.fit.factory.JCasFactory;
import org.apache.uima.fit.pipeline.SimplePipeline;
import de.tudarmstadt.ukp.dkpro.core.api.segmentation.type.*;
import de.tudarmstadt.ukp.dkpro.core.api.syntax.type.*;
import static org.apache.uima.fit.util.JCasUtil.*;
import static org.apache.uima.fit.factory.AnalysisEngineFactory.*;

def jcas = JCasFactory.createJCas();
jcas.documentText = "This is a test";
jcas.documentLanguage = "en";

SimplePipeline.runPipeline(jcas,
  createEngineDescription(OpenNlpSegmenter),
  createEngineDescription(OpenNlpPosTagger),
  createEngineDescription(OpenNlpParser,
    OpenNlpParser.PARAM_WRITE_PENN_TREE, true));

select(jcas, Token).each { println "${it.coveredText} ${it.pos.posValue}" }
select(jcas, PennTree).each { println it.pennTree }
```

Fetches all required dependencies

No manual installation!

Input

Analytics pipeline. Language-specific resources fetched automatically

Output

DKPro Script – Groovy-based DSL

```
#!/usr/bin/env groovy
import groovy.transform.BaseScript
@Grab('org.dkpro.core:dkpro-core-groovy:1.0.0-SNAPSHOT')
@BaseScript DKProCoreScript baseScript
```

Fetches all required dependencies

No manual installation!

Why is this cool?

Domain-specific Language built with Groovy

Still a Groovy program, but syntactic sugar + pre-configuration

```
read 'String' language 'de' pa
        documentText: 'This is
```

Input

```
apply 'OpenNlpSegmenter'
apply 'OpenNlpPosTagger'
apply 'OpenNlpParser' params([
        writePennTree: true])
```

Analytics pipeline. Language-specific resources fetched automatically

```
write 'CasDump'
```

Output

Built-in help

- List ‘inventory’
- ‘explain’ components and formats

Work in progress…

https://dkpro.github.io/dkpro-script

IT'S ALL ABOUT THE METADATA

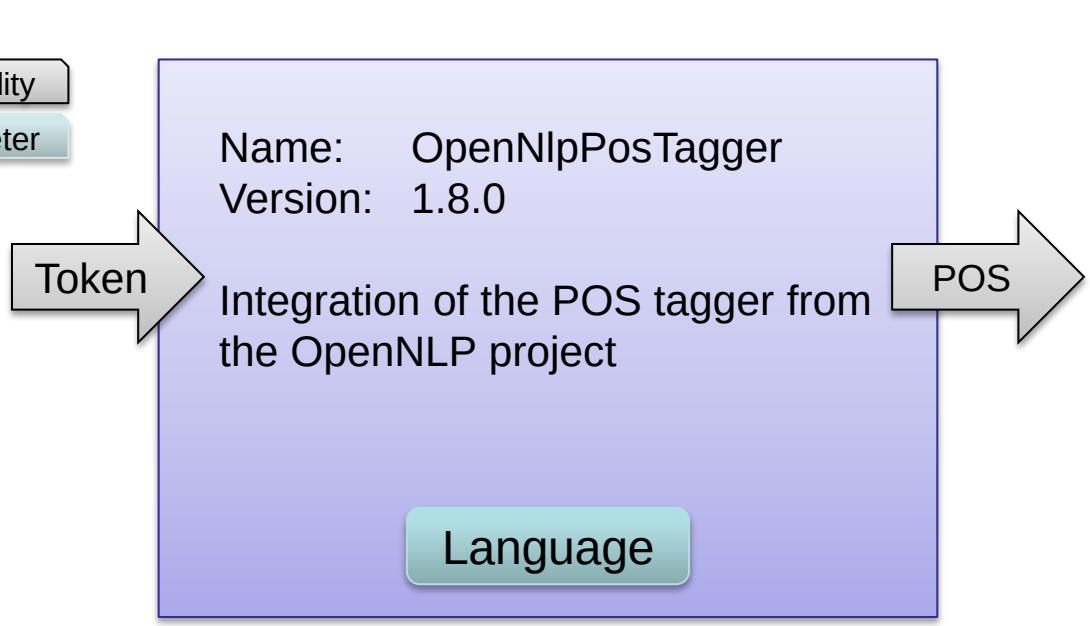
Exploiting metadata

TECHNISCHE UNIVERSITÄT DARMSTADT

- DKPro Core incorporates metadata on many levels
  - Components
  - Models
  - Type system
  - Datasets
  - Formats
  - Tagsets
- … from many sources and different formats
  - Java source code (e.g., JavaDoc, Java annotations)
  - Maven project descriptions
  - Ant build files
  - Java properties files
  - ...

Analysis Engine Descriptor

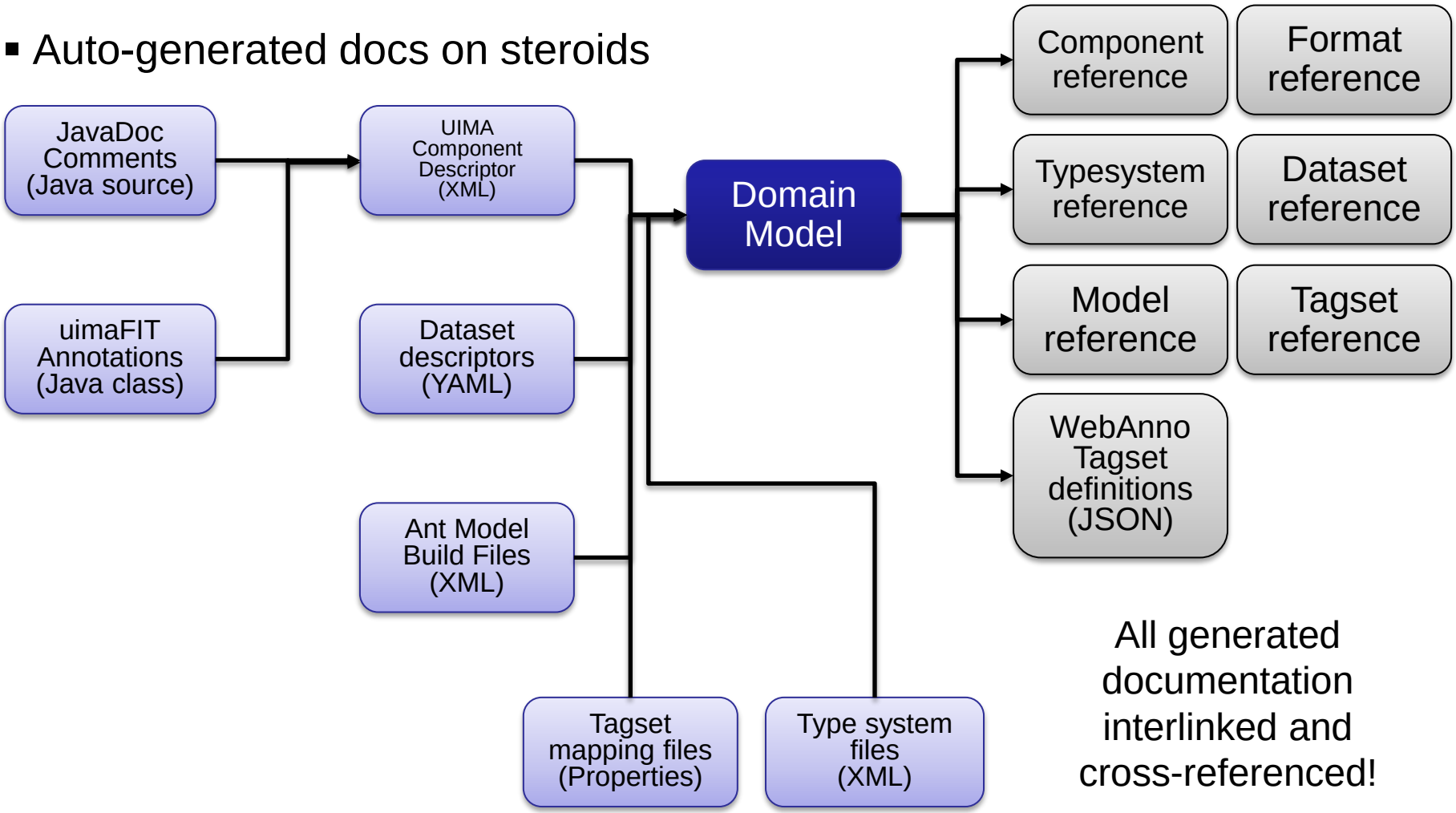
- Name
- Version
- Vendor
- Type system
- Parameters
- Capabilities
- Indexes
- Resources
- Single- / multiple deployment
- Delegate Analysis Engines (aggregate AEs only)
- Flow control (aggregate AEs only)
- … a few more



Exploiting metadata

DKPro Core Reference Documentation

- Auto-generated docs on steroids



All generated documentation interlinked and cross-referenced!

Exploiting metadata

OpenMinTeD Component Overview

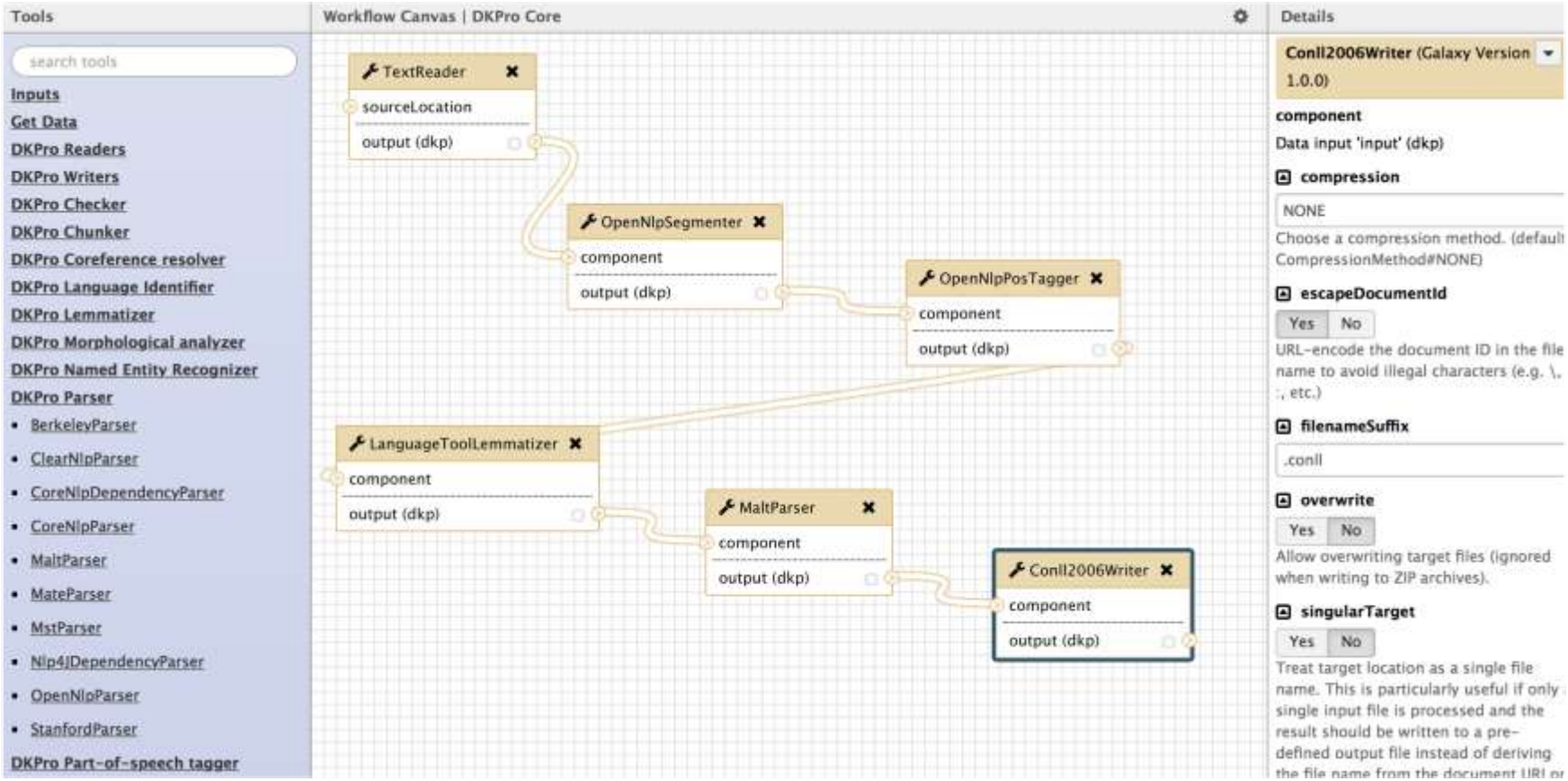
(original) GATE (135)

The components listed here could not be associated with a known third-party tool collection and are assumed to be original components.

Component	Description	Framework
ANNIE English Tokeniser	A customisable English tokeniser.	GATE
ANNIE Gazetteer	A list lookup component.	GATE
ANNIE NE Transducer	ANNIE named entity grammar.	GATE
ANNIE Nominal Coreferencer	Nominal Coreference resolution component	GATE
ANNIE OrthoMatcher	ANNIE orthographical coreference component.	GATE
ANNIE Pronominal	Pronominal Coreference	GATE

Exploiting metadata

Generating Galaxy Tool Wrappers



Source: Thesis presentation Tahir Hussain

dkprocore

What comes next?

dkprocore

Questions?

THANKS!

https://dkpro.github.io/dkpro-core

Image credits

- TU Darmstadt S103 ErhoehtVonS208 © 2007 ThomasGP. CC BY-SA 4.0.
- Robert-Piloty-Gebäude, TU Darmstadt © 2006 S. Kasten. CC BY-SA 4.0.
- Darmstadt 2006 121 © 2006 derbrauni. CC BY-SA 4.0.
- Darmstadt TU 1 © 2011 Andreas Pfaefcke. CC BY 3.0.
- University College Front Facade © 2004 Nuthingoldstays. CC BY-SA 3.0.
- First Nations University 3 © 2013 Nadiatalent. CC BY-SA 3.0.
- LogoJava.png by Christian F. Burprich, CC BY-NC-SA 3.0
- LogoPython.png by IFA
- LogoGroovy.png by pictonic.co
- IconComponents.png, IconModels.png by Visual Pharm
- IconFormatText.png, IconFormatBlank.png by Honza Dousek
- IconTypeSystem.png by Designmodo