UIMA Tutorial – uimaFIT & DKPro Core

TECHNISCHE UNIVERSITÄT DARMSTADT

3rd UIMA@GSCL Workshop, GSCL 2013, Darmstadt

Agenda

- What is a pipeline?
- Working with annotations
  - What is a type system?
  - What is the Common Analysis Structure (CAS)?
- Working with components
  - What is a reader?
  - What is an analysis engine?
  - What is a writer? (aka consumer)
- DKPro Core component collection

Learning to read is difficult …

TECHNISCHE UNIVERSITÄT DARMSTADT

$�⊢�‡→○⏏%⊢ $%$�▷̇⊥ →○▷̇↕ ♦▷̇%¥ ⊥↕%б↕ -- $▷̇⁖∇б▷̇ ↕♦▷̇] #%‡ ↕%⊢⁂
▷̇⁑▷̇‡ $▷̇⁖∇б▷̇ ↕♦▷̇] #%‡ $%$$⊢▷̇ % ⊥�‡→○⊢▷̇ ◗∇б¥, $%$�▷̇⊥ �‡
$�⊢�‡→○⏏%⊢ ♦∇⏏⊥▷̇♦∇⊢¥⊥ =%] →○▷̇↕ % ♦▷̇%¥ ⊥↕%б↕ �‡ ⊢�⁖▷̇, %##∇б
¥�‡→○ ↕∇ % ↕▷̇%= ∇⁖ ⊥#�▷̇‡↕�⊥↕⊥ �‡ �↕%⊢]. б%↕♦▷̇б ↕♦%‡
#∇‡⁖⏏⊥�‡→○ $%$�▷̇⊥, ♦▷̇%б�‡→○ =∇б▷̇ ↕♦%‡ ∇‡▷̇ ⊢%‡→○⏏%→○▷̇
→○�⁑▷̇⊥ ‡▷̇◗$∇б‡⊥ % =▷̇‡↕%⊢ $∇∇⊥↕, %##∇б¥�‡→○ ↕∇ ↕♦▷̇ ‡▷̇◗ ⊥↕⏏¥],
◗♦�#♦ ↕▷̇⊥↕▷̇¥ ⊥▷̇⁑▷̇‡-=∇‡↕♦-∇⊢¥ �‡⁖%‡↕⊥ "�‡ =%‡] ▷̇⏏б∇◦▷̇%‡
#∇⏏‡↕б�▷̇⊥, ◦%б▷̇‡↕⊥ %б▷̇ ◗%б] ∇⁖ →○�⁑�‡→○ % $�⊢�‡→○⏏%⊢ ▷̇¥⏏#
%↕�∇‡ ↕∇ ↕♦▷̇�б ⁂�¥⊥ %‡¥ ↕б] ↕∇ ⊥◦▷̇%⁂ ∇‡⊢] ∇‡▷̇ ⊢%‡→○⏏%→○▷̇," ⊥
%�¥ ⊥↕⏏¥] %⏏↕♦∇б ↗%#⁋⏏▷̇⊥ =▷̇♦⊢▷̇б ∇⁖ ↕♦▷̇ ⊢%‡→○⏏%→○▷̇,
#∇→○‡�↕�∇‡, %‡¥ ¥▷̇⁑▷̇⊢∇◦=▷̇‡↕ ⊢%$ %↕ ↕♦▷̇ �‡↕▷̇б‡%↕�∇‡%⊢ ⊥#♦∇∇⊢
⁖∇б %¥⁑%‡#▷̇¥ ⊥↕⏏¥�▷̇⊥ �‡ ↕б�▷̇⊥↕▷̇, �↕%⊢]. "↕♦▷̇] %б▷̇ %⁖б%�¥
[↕♦▷̇�б #♦�⊢¥б▷̇‡] =�→○♦↕ ⊥⏏⁖⁖▷̇б ◗♦▷̇‡ ↕♦▷̇] →○▷̇↕ ↕∇ ⊥#♦∇∇⊢ %‡¥ ⊥∇
∇‡," =▷̇♦⊢▷̇б ⊥%�¥. "$▷̇#%⏏⊥▷̇ ∇⁖ ∇⏏б б▷̇⊥⏏⊢↕⊥, � ¥∇⏏$↕ ↕♦%↕ ⁑▷̇б]
=⏏#♦."

What is UIMA?

- Component-based architecture for analysis of unstructured data
- „Analysis“ means deriving a structure from the unstructured data

- How does it work?

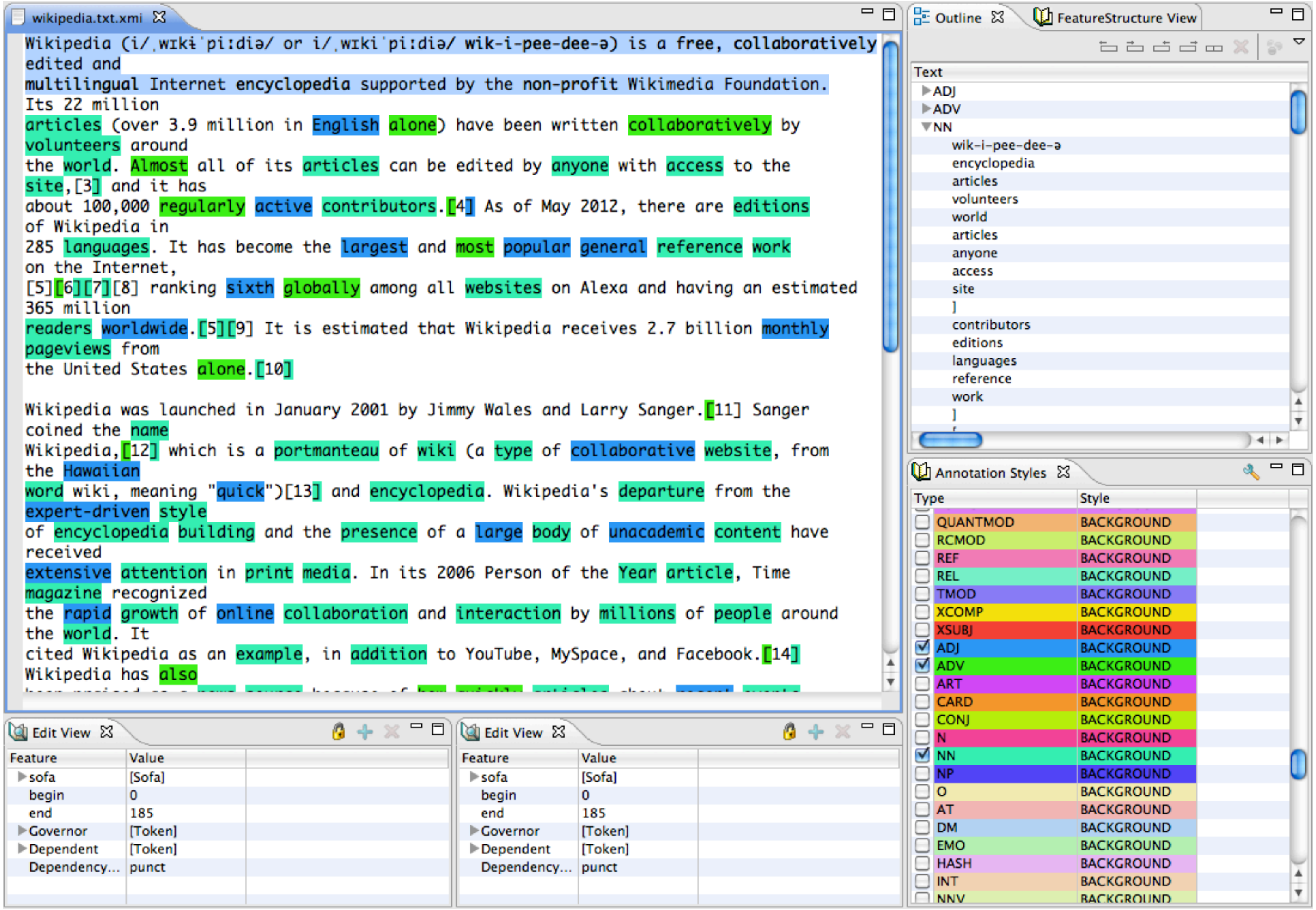
Like an assembly line...

Take the raw material

Refine it step by step

Drive off with a nice car

Output Example (UIMA Annotation Editor)



Wikipedia (i/ˌwɪkɨˈpiːdiə/ or i/ˌwɪkiˈpiːdiə/ wik-i-pee-dee-ə) is a free, collaboratively edited and
multilingual Internet encyclopedia supported by the non-profit Wikimedia Foundation. Its 22 million
articles (over 3.9 million in English alone) have been written collaboratively by volunteers around
the world. Almost all of its articles can be edited by anyone with access to the site,[3] and it has
about 100,000 regularly active contributors.[4] As of May 2012, there are editions of Wikipedia in
285 languages. It has become the largest and most popular general reference work on the Internet,
[5][6][7][8] ranking sixth globally among all websites on Alexa and having an estimated 365 million
readers worldwide.[5][9] It is estimated that Wikipedia receives 2.7 billion monthly pageviews from
the United States alone.[10]

Wikipedia was launched in January 2001 by Jimmy Wales and Larry Sanger.[11] Sanger coined the name
Wikipedia,[12] which is a portmanteau of wiki (a type of collaborative website, from the Hawaiian
word wiki, meaning "quick")[13] and encyclopedia. Wikipedia's departure from the expert-driven style
of encyclopedia building and the presence of a large body of unacademic content have received
extensive attention in print media. In its 2006 Person of the Year article, Time magazine recognized
the rapid growth of online collaboration and interaction by millions of people around the world. It
cited Wikipedia as an example, in addition to YouTube, MySpace, and Facebook.[14] Wikipedia has also

Text

- ADJ
- ADV
- NN
  - wik-i-pee-dee-ə
  - encyclopedia
  - articles
  - volunteers
  - world
  - articles
  - anyone
  - access
  - site
  -]
  - contributors
  - editions
  - languages
  - reference
  - work
  -]

Type	Style
□ QUANTMOD	BACKGROUND
□ RCMOD	BACKGROUND
□ REF	BACKGROUND
□ REL	BACKGROUND
□ TMOD	BACKGROUND
□ XCOMP	BACKGROUND
□ XSUBJ	BACKGROUND
☑ ADJ	BACKGROUND
☑ ADV	BACKGROUND
□ ART	BACKGROUND
□ CARD	BACKGROUND
□ CONJ	BACKGROUND
□ N	BACKGROUND
☑ NN	BACKGROUND
□ NP	BACKGROUND
□ O	BACKGROUND
□ AT	BACKGROUND
□ DM	BACKGROUND
□ EMO	BACKGROUND
□ HASH	BACKGROUND
□ INT	BACKGROUND
□ NNV	BACKGROUND

Edit View

Feature	Value
sofa	[Sofa]
begin	0
end	185
Governor	[Token]
Dependent	[Token]
Dependency...	punct

Edit View

Feature	Value
sofa	[Sofa]
begin	0
end	185
Governor	[Token]
Dependent	[Token]
Dependency...	punct

Apache UIMA™ - Some history

Unstructured Information Management Architecture

- 2003 – David Ferrucci and Adam Lally paper
  Accelerating corporate research in the development, application and deployment of human language technologies
- 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

http://uima.apache.org

Important features of UIMA

- Common representation format for annotations
- Customizable annotation type system
- Common API for analysis components
- Focus on the ability to scale

Past:	UIMA-CPE	– Collection Processing Engine
Present:	UIMA-AS	– Asynchronous Scale-out
Future:	UIMA-DUCC	– Distributed UIMA Cluster Computing

- Active community

Apache uimaFIT™

- 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

- Philip V. Ogren, Steven J. Bethard (2009) **Building Test Suites for UIMA Components.** Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing (SETQA-NLP 2009). June 2009.
- Christophe Roeder, Philip V. Ogren, William A. Baumgartner Jr., Lawrence Hunter (2009). **Simplifying UIMA Component Developmnet and Testing with Java Annotations and Dependency Injection**, in Chiarcos, C., Eckhart de Castilho, Stede, M. (eds), Von der Form zur Bedeutung: Text automatisch verarbeiten / From Form to Meaning: Processing Texts Automatically. Tübingen: Narr, 2009

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

Important features of uimaFIT

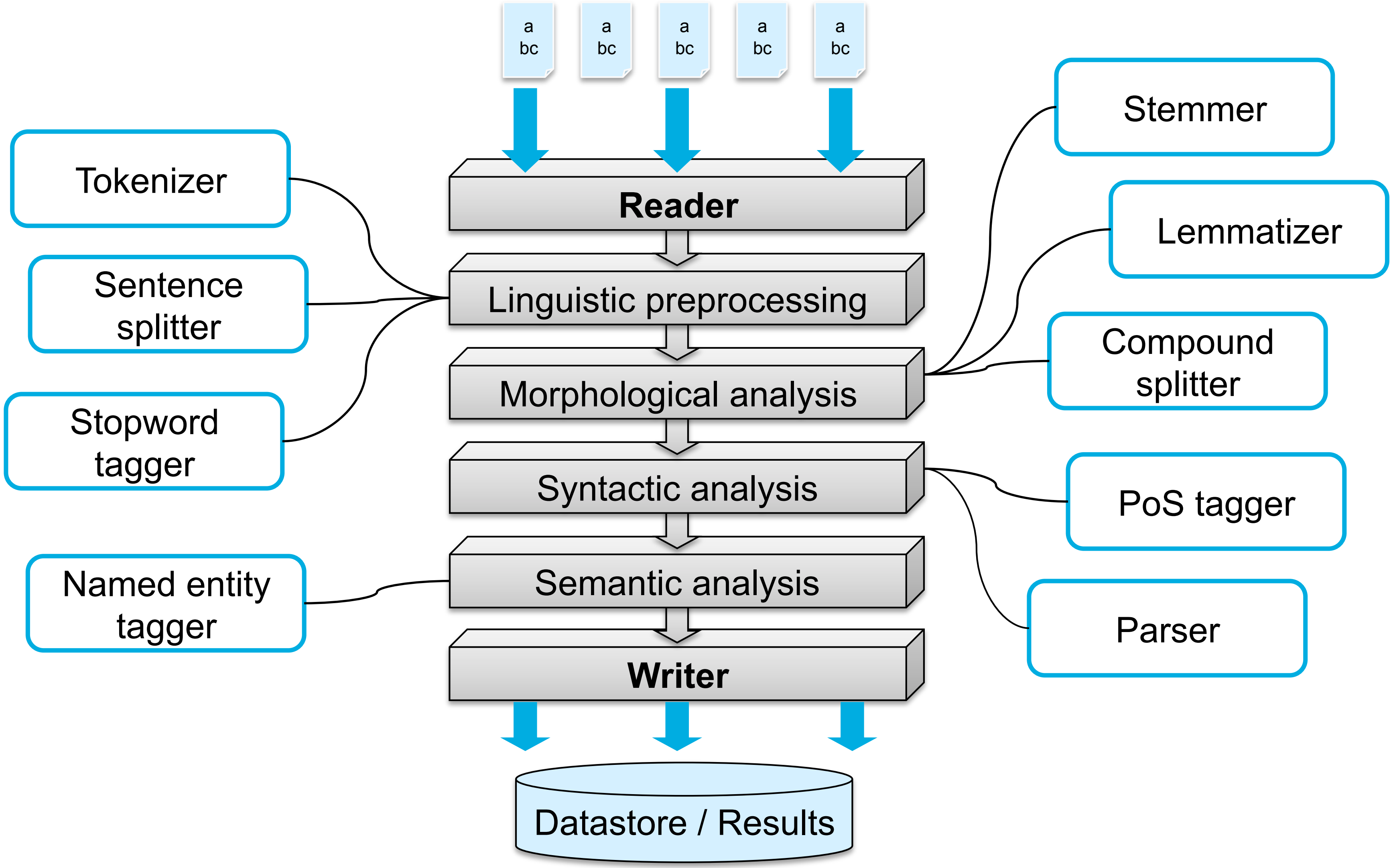
uimaFIT is key to make UIMA usable within Java code

- **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

… and more …

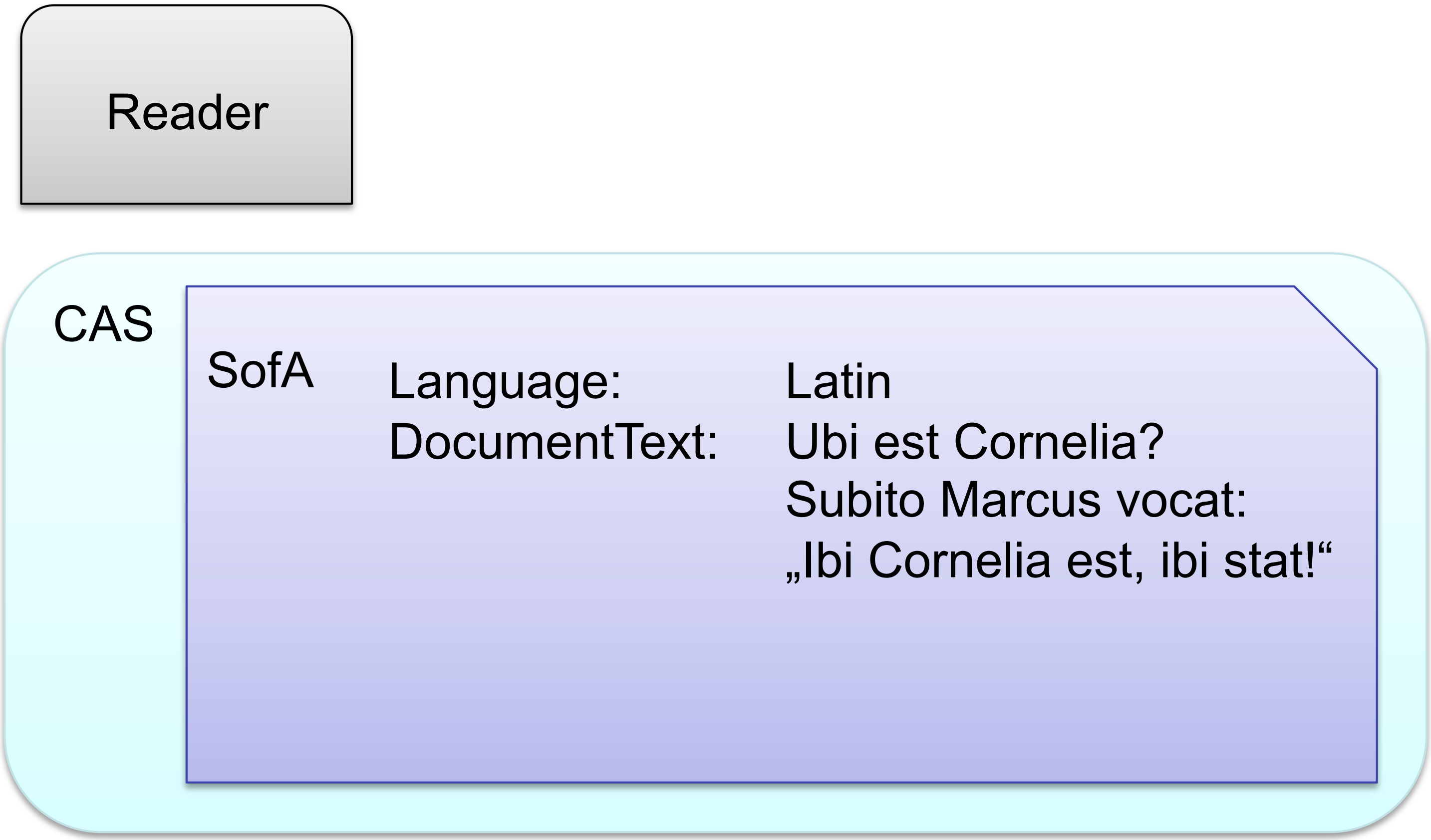
Pipelines

Pipeline Architecture



Component – Collection Reader

- Empty data structure (CAS) is passed to the reader
- Reader sets text (SofA) and meta-data (e.g. language)



Reader

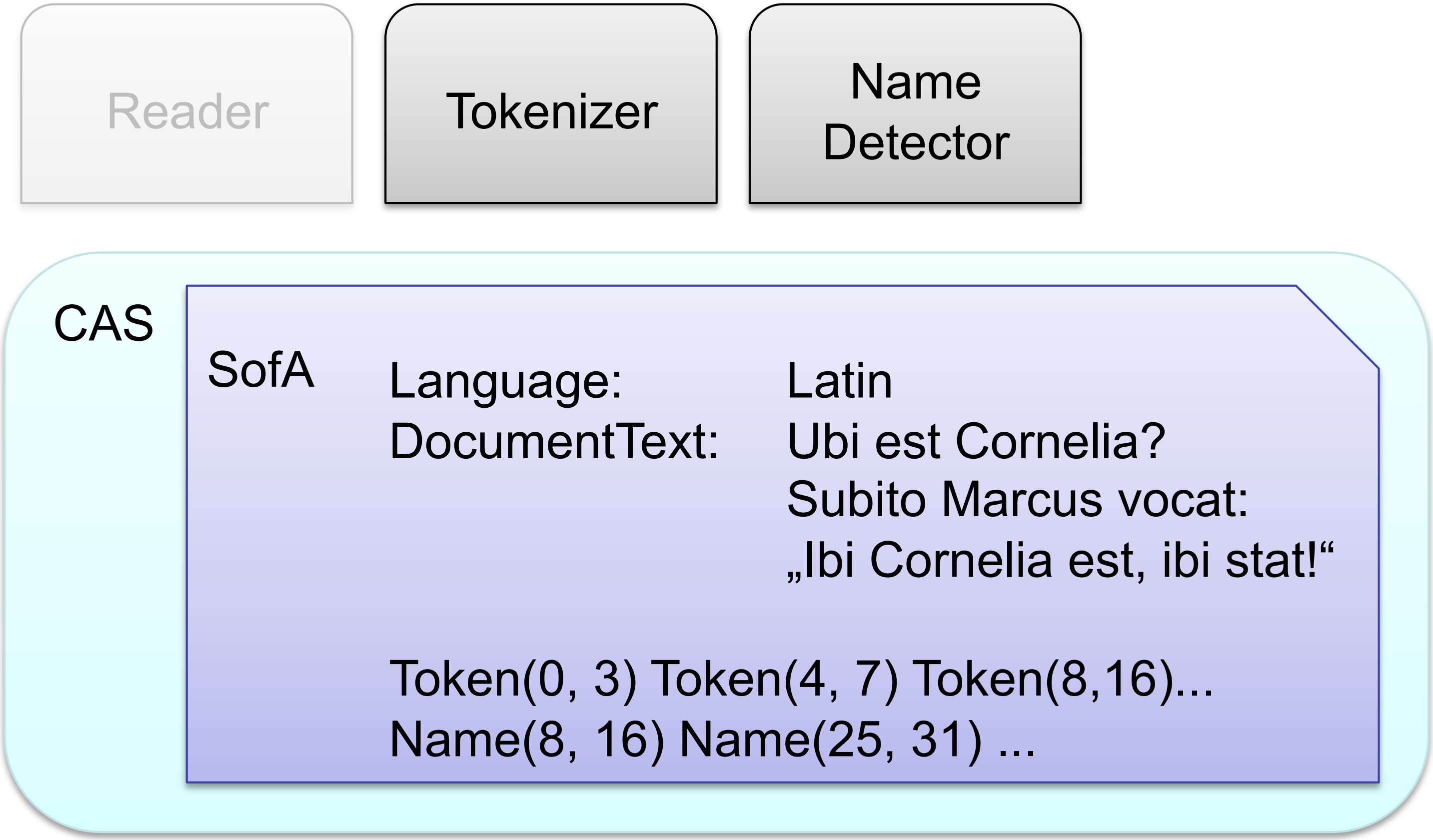
CAS

SofA

Language:	Latin
DocumentText:	Ubi est Cornelia? Subito Marcus vocat: „Ibi Cornelia est, ibi stat!“

Component – Analysis Engine

- The structure is passed to one analysis engine (AE) after the other
- Each analysis engine derives a bit of structure and records it (Annotation)



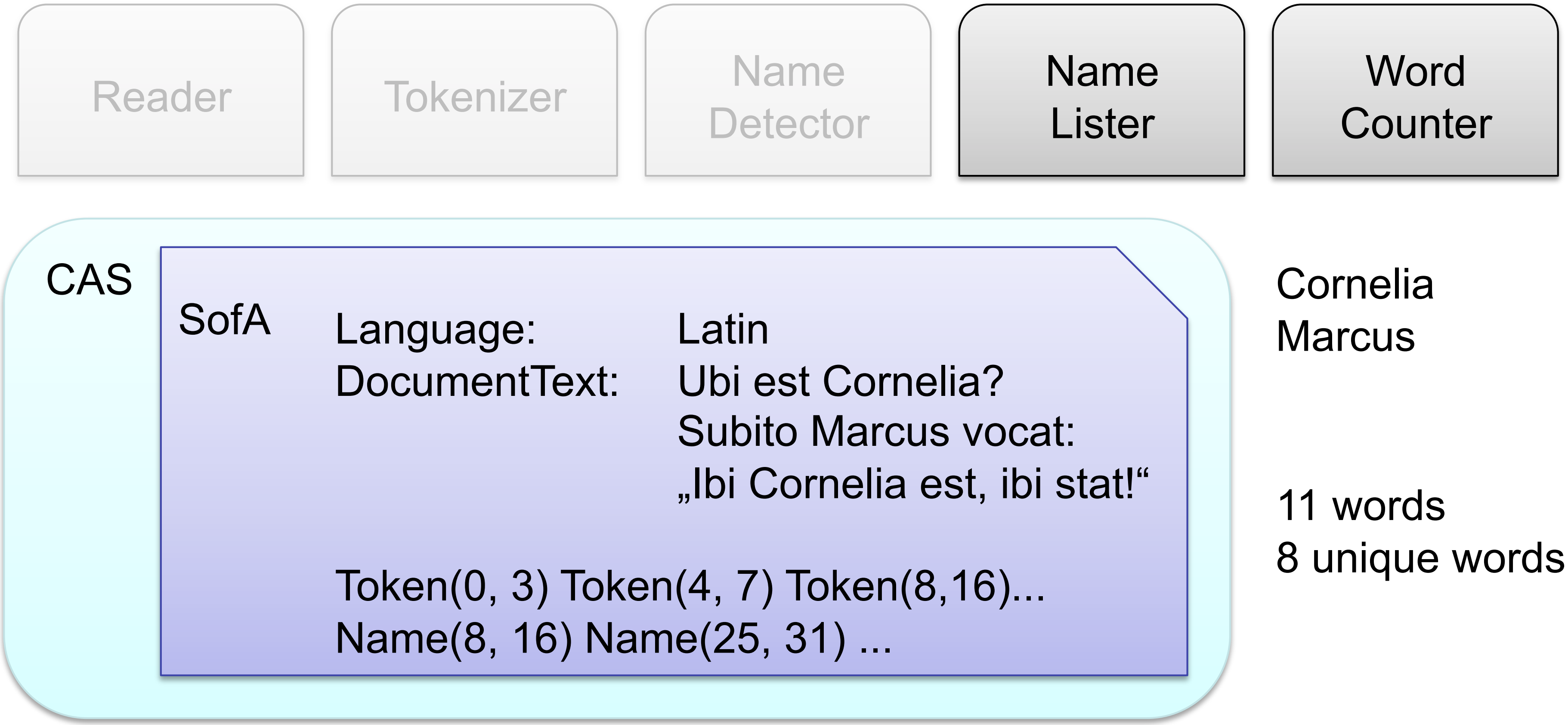
Language: Latin

DocumentText: Ubi est Cornelia? Subito Marcus vocat: „Ibi Cornelia est, ibi stat!“

Token(0, 3) Token(4, 7) Token(8,16)...
Name(8, 16) Name(25, 31) ...

Component – CAS Consumer

- Do something interesting with the analyzed data



Reader

Tokenizer

Name Detector

Name Lister

Word Counter

CAS

SofA

Language: Latin

DocumentText: Ubi est Cornelia? Subito Marcus vocat: „Ibi Cornelia est, ibi stat!“

Token(0, 3) Token(4, 7) Token(8,16)...

Name(8, 16) Name(25, 31) ...

Cornelia
Marcus

11 words
8 unique words

Pipeline Example

```
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”);
```

UIMA Data Structures

Common Analysis System (CAS)

- Provides access to primary data
- Stores secondary data aka annotations
- Functions like an in-memory database
  - Annotation types are like “tables”
  - There are “indexes”

Type System

- UIMA specification is platform-independent
- Cannot rely on type system of implementation language (Java, C++)
- UIMA provides an “Object-oriented” type-system with
  - Type -> class
  - Feature -> class member
  - Feature Structure -> instance
  - Single inheritance
  - Sub-type polymorphism
  - no methods or encapsulation
- Primitive types: integer, float, boolean, string
- Built-in complex types: arrays, lists, Annotation
- Type-system forms communication contract between components

Java + CAS = JCas

- JCas maps CAS types into the Java type system
- JCasGen generates Java classes from the XML type system descriptor
  - Token.java – feature structure wrapper with getters and setters
  - Token_type.java – type wrapper (cf. Java ‘Class’ class)
- JCas wrappers cannot be used stand-alone
- Type system descriptors still needed to initialize the underlying CAS

Code: CAS (UIMA)

File: CasAndJCasExample

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

CAS cas = CasCreationUtils.createCas(tsd, null, null);
cas.setDocumentText("This is a test.");

Type tokenType = cas.getTypeSystem().getType("Token");
cas.addFsToIndexes(cas.createAnnotation(tokenType, 0, 4));
cas.addFsToIndexes(cas.createAnnotation(tokenType, 5, 7));
cas.addFsToIndexes(cas.createAnnotation(tokenType, 8, 9));
cas.addFsToIndexes(cas.createAnnotation(tokenType, 10, 14));
cas.addFsToIndexes(cas.createAnnotation(tokenType, 14, 15));

Feature lengthFeat = tokenType.getFeatureByBaseName("length");
AnnotationIndex<AnnotationFS> tokenIdx = cas.getAnnotationIndex(tokenType);
for (AnnotationFS token : tokenIdx) {
  token.setIntValue(lengthFeat, token.getCoveredText().length());
}

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

Code: JCas (uimaFIT)

File: CasAndJCasExample

```
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());
}
```

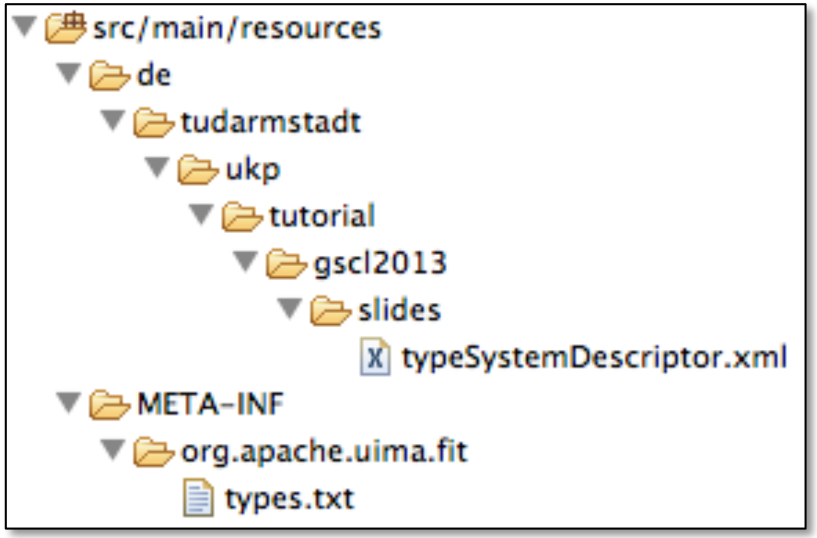
Navigating the CAS with JCasUtil/CasUtil

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

```
// CAS version
Type tokenType = CasUtil.getType(cas, "my.Token");
for (AnnotationFS token : CasUtil.select(cas, tokenType)) {
  ...
}

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

uimaFIT type system detection

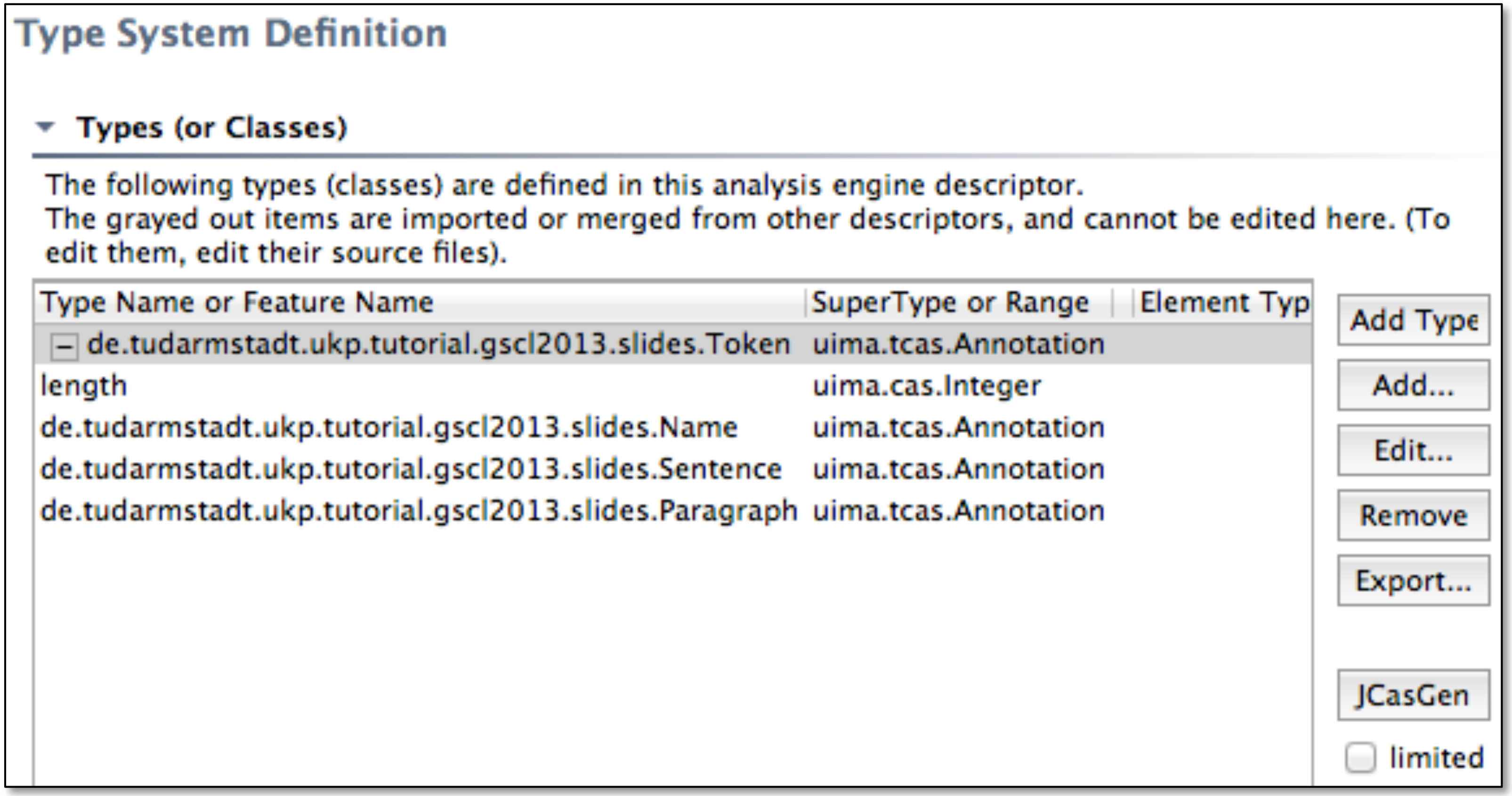
- uimaFIT example doesn’t explicitly load/create type system
- Type system detection mechanism
- Types defined in XML descriptor files
- uimaFIT scans classpath for type system descriptor files



```
▼ src/main/resources
  ▼ de
    ▼ tudarmstadt
      ▼ ukp
        ▼ tutorial
          ▼ gscl2013
            ▼ slides
                typeSystemDescriptor.xml
  ▼ META-INF
    ▼ org.apache.uima.fit
        types.txt
```

Type System Editor (Eclipse)

File: typeSystemDescriptor.xml (gscl2013-slides)



Type System Definition

Types (or Classes)

The following types (classes) are defined in this analysis engine descriptor.
The grayed out items are imported or merged from other descriptors, and cannot be edited here. (To edit them, edit their source files).

Type Name or Feature Name	SuperType or Range	Element Typ
de.tudarmstadt.ukp.tutorial.gscl2013.slides.Token	uima.tcas.Annotation	
length	uima.cas.Integer	
de.tudarmstadt.ukp.tutorial.gscl2013.slides.Name	uima.tcas.Annotation	
de.tudarmstadt.ukp.tutorial.gscl2013.slides.Sentence	uima.tcas.Annotation	
de.tudarmstadt.ukp.tutorial.gscl2013.slides.Paragraph	uima.tcas.Annotation	

- JCasGen makes UIMA types available as Java Classes

Type Priorities

In which order does select() return annotations?

- Type priorities define iteration order over features structures
- Example
  - A Sentence and a Paragraph start and end at the same position
- Which should be returned first?
  - Paragraph conceptually „larger“, but UIMA cannot know that by itself
- User specifies type priority: Paragraph, Sentence

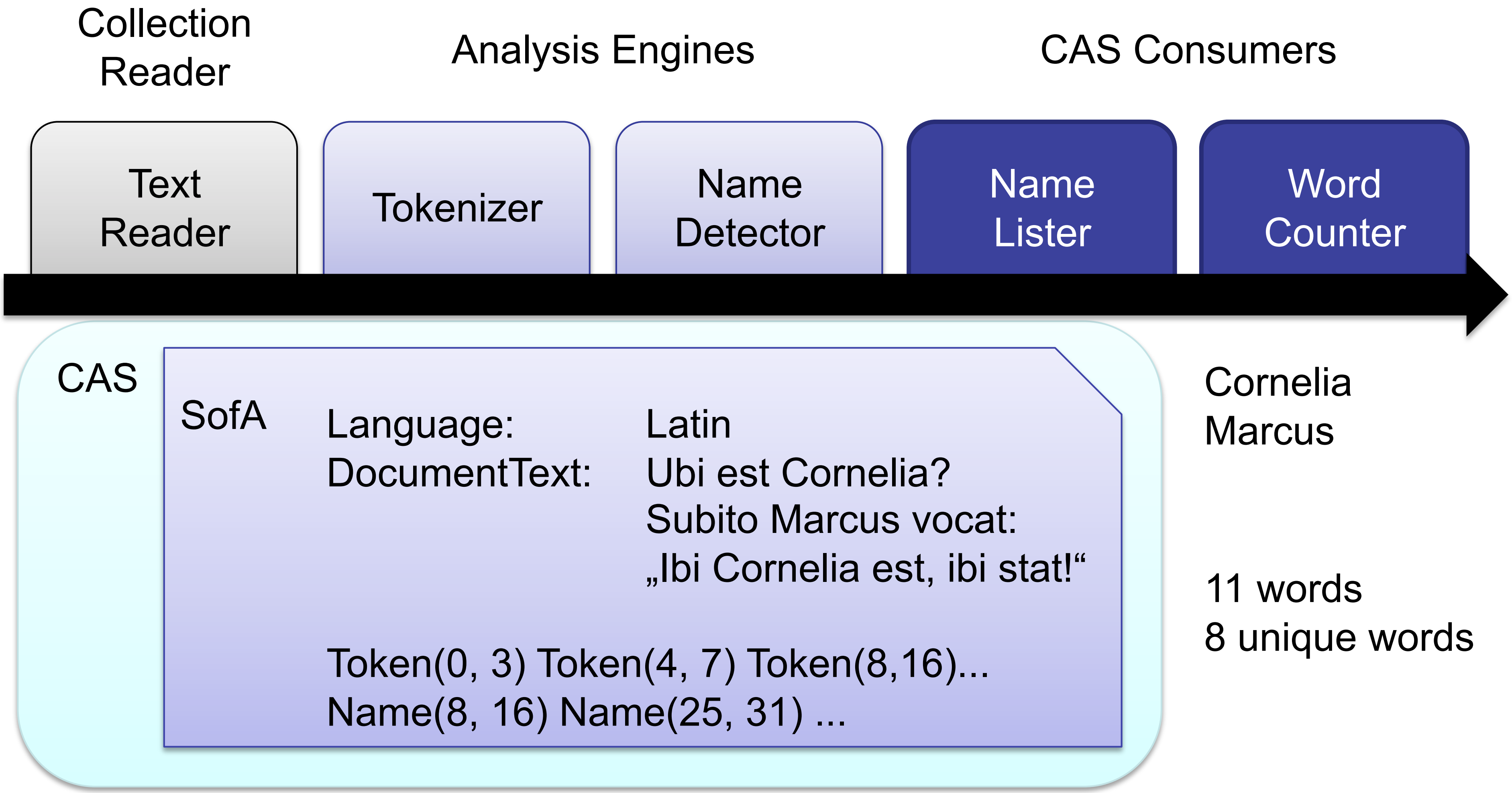
Code: Type Priorities (JCas + uimaFIT)

File: TypePrioritiesExample

```
TypePriorities prio = createTypePriorities(
  Paragraph.class, Sentence.class, Token.class);

JCas jcas = createCas(createTypeSystemDescription(), prio,
  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();
new Sentence(jcas, 0, 15).addToIndexes();
new Paragraph(jcas, 0, 15).addToIndexes();

for (Annotation a : select(jcas, Annotation.class)) {
  System.out.println("[" + a.getType().getShortName() + "|" +
    a.getBegin() + "-“ + a.getEnd() + "] " + a.getCoveredText());
}
```

Components

Components



Collection Reader

Analysis Engines

CAS Consumers

Text Reader

Tokenizer

Name Detector

Name Lister

Word Counter

CAS

SofA

Language: Latin

DocumentText: Ubi est Cornelia? Subito Marcus vocat: „Ibi Cornelia est, ibi stat!“

Token(0, 3) Token(4, 7) Token(8,16)...
Name(8, 16) Name(25, 31) ...

Cornelia
Marcus

11 words
8 unique words

API – Life-Cycle Events

- Component life-cycle events
  - initialize()
    - configure component from context
    - apply parameters, initialize resources
  - reconfigure()
    - context has changed
    - re-initialize component
  - destroy()
    - release resources
- Processing life-cycle events
  - collectionProcessComplete()
    - last CAS in collection has been processed
    - aggregated analysis complete
  - batchProcessComplete()
    - last CAS in batch has been processed
    - recoverable components set checkpoint
- Other
  - typeSystemInit() *(CASAnnotator_ImplBase only)*
    - type-system has changed

API – Processing Methods

- CollectionReader
  - hasNext() - another CAS can be filled
  - getNext() - fill the next CAS
  - getProgress() - report current progress to execution engine

- AnalysisEngine
  - process() - process/modify CAS

- CasConsumer
  - process() - process CAS; do not modify

Code: initialize() (uimaFIT)

File: TextFileReader

```
public static final String PARAM_PATH = "path";
@ConfigurationParameter(name = PARAM_PATH, mandatory = true)
protected File path;

public static final String PARAM_FILENAME_PATTERN = "filenamePattern";
@ConfigurationParameter(name = PARAM_FILENAME_PATTERN, mandatory = true,
  defaultValue = ".*\\.txt")
protected String filenamePattern;

public static final String PARAM_LANGUAGE = "language";
@ConfigurationParameter(name = PARAM_LANGUAGE, mandatory = true)
protected String language;

protected Queue<File> files;
protected int totalFiles;

public void initialize(UimaContext aContext) throws
ResourceInitializationException {
  super.initialize(aContext);
  files = new LinkedList<File>();
  collectFiles(path, filenamePattern, files);
  totalFiles = files.size();
}
```

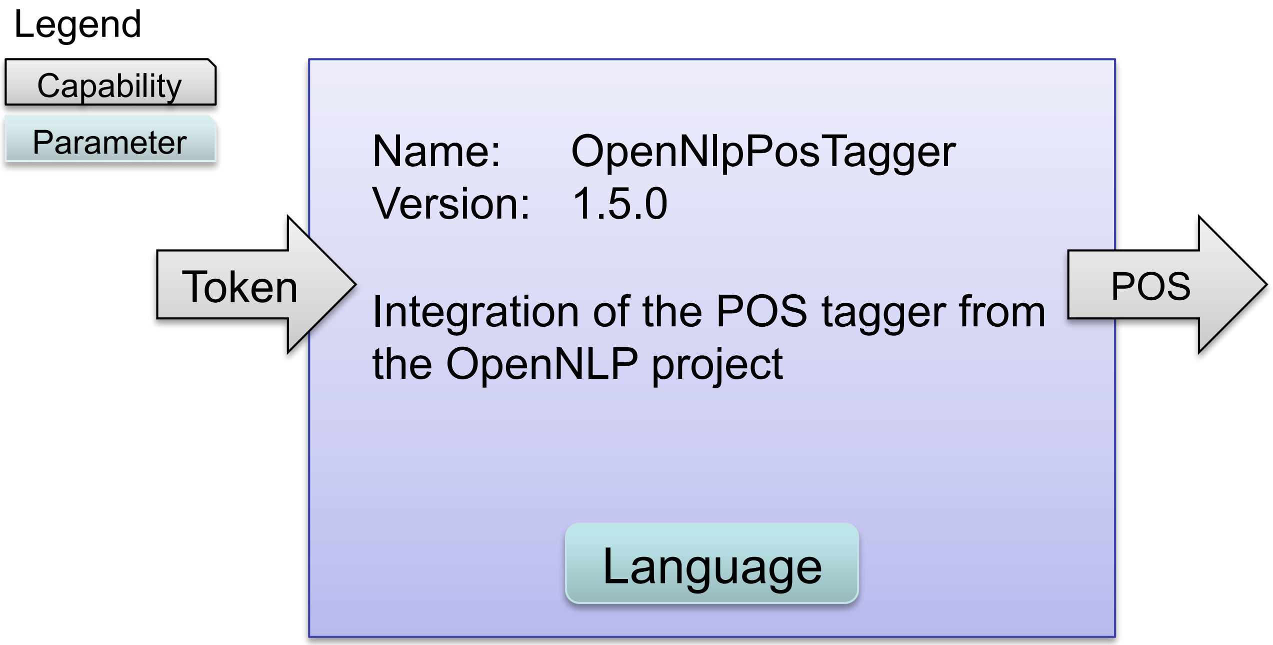
Code: process() (uimaFIT)

File: NameAnnotator

```
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();
    }
  }
}
```

Figure: 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



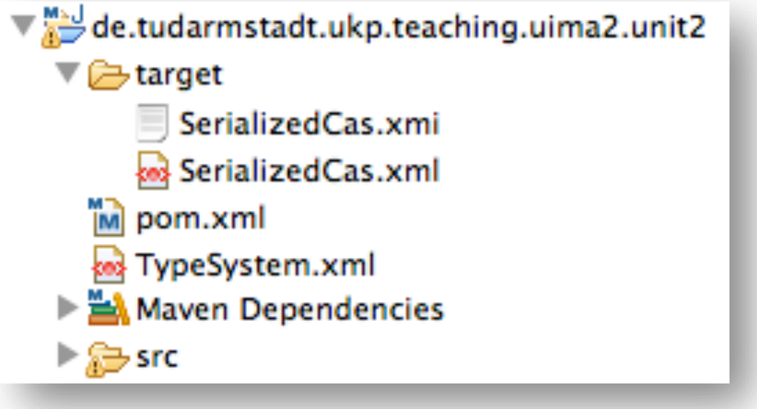
XML Descriptors – Pro & Contra

- Pro
  - “Officially preferred” form of configuring UIMA components/resources
  - Widely supported by UIMA tooling
  - XML elements usually correspond 1:1 to Java classes

- Contra
  - Mix declaration/documentation and configuration
  - Not included when refactoring code
  - No convenient API for use in Java (remedy: uimaFIT factories)

Persisting and loading a CAS

- Available serialization formats
  - XCAS – proprietary UIMA XML format (rarely used today)
  - XMI – standard XML representation of object graphs
- Type-system definition not included!



```
▼ de.tudarmstadt.ukp.teaching.uima2.unit2
  ▼ target
      SerializedCas.xmi
      SerializedCas.xml
    pom.xml
    TypeSystem.xml
  ▶ Maven Dependencies
  ▶ src
```

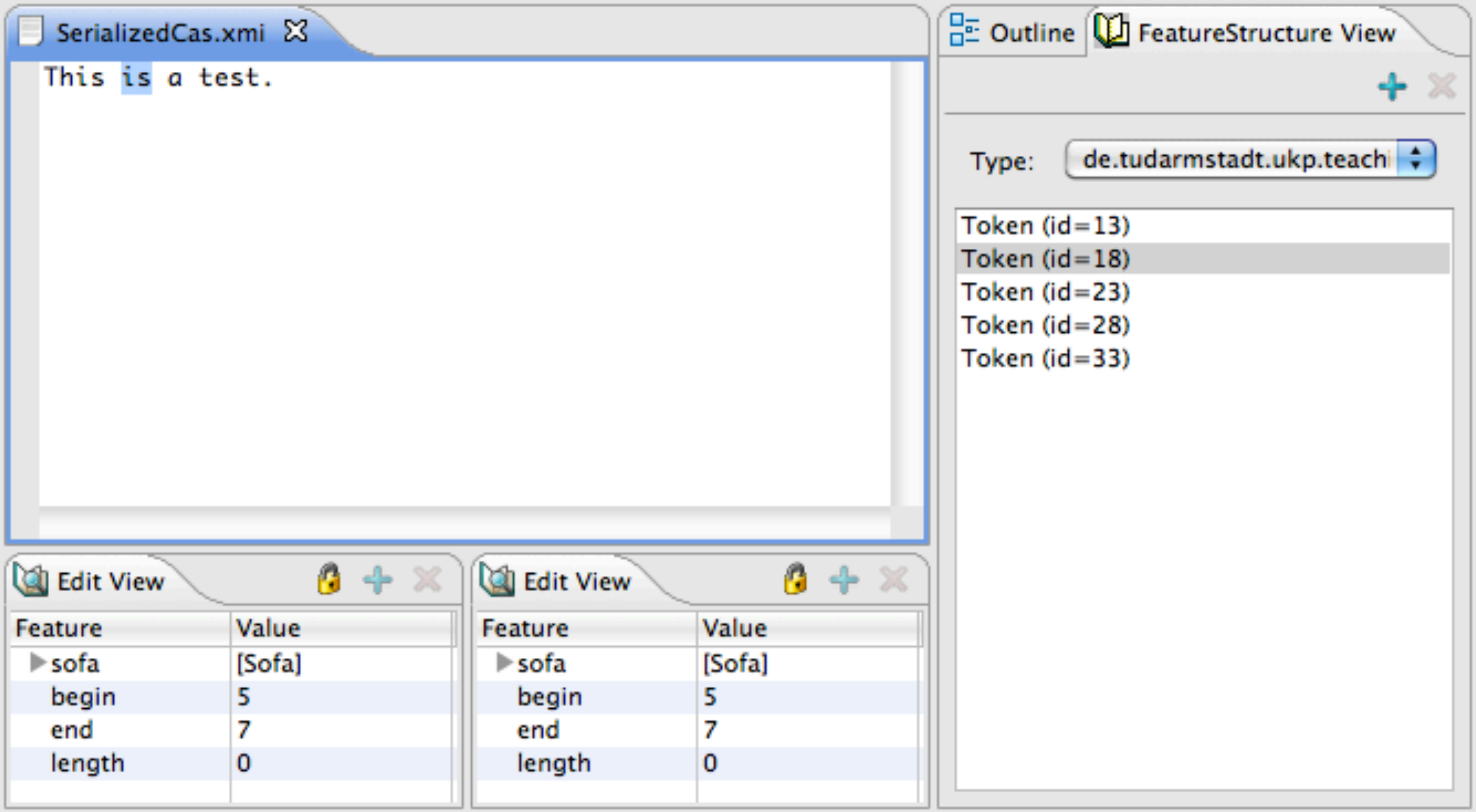
- Tip
  - Persist type system as “TypeSystem.xml” at project root
  - Open and XMI file in that project with the CAS Editor

Code: CAS to XMI (de)-serialization

File: CasPersistenceExample

```
CAS cas = createCas(createTypeSystemDescription(), null, null);
populateCas(cas);

FileOutputStream out = new FileOutputStream("target/SerializedCas.xmi");
XmiCasSerializer.serialize(cas, out);
closeQuietly(out);

CAS loadedCas = createCas(createTypeSystemDescription(), null, null);
FileInputStream in = new FileInputStream("target/SerializedCas.xmi");
XmiCasDeserializer.deserialize(in, loadedCas);
closeQuietly(in);

createPrimitive(PrintConsumer.class).process(loadedCas);

FileOutputStream typeOut = new FileOutputStream("TypeSystem.xml");
createTypeSystemDescription().toXML(typeOut);
closeQuietly(typeOut);
```

CAS Editor



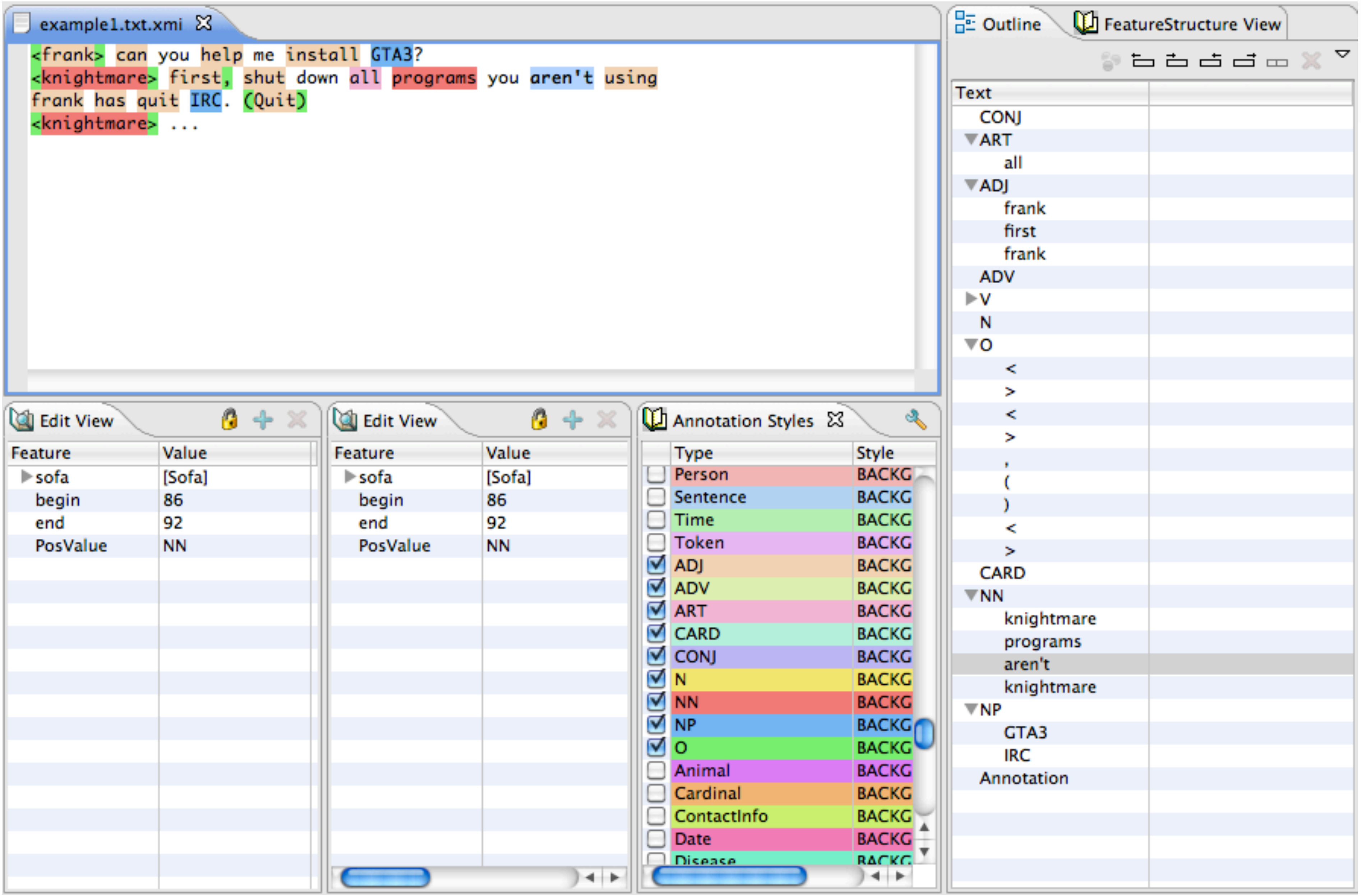
```
This is a test.
```

Type: de.tudarmstadt.ukp.teachi

Token (id=13)
Token (id=18)
Token (id=23)
Token (id=28)
Token (id=33)

Feature	Value
▶ sofa	[Sofa]
begin	5
end	7
length	0

Feature	Value
▶ sofa	[Sofa]
begin	5
end	7
length	0



```
<frank> can you help me install GTA3?
<knightmare> first, shut down all programs you aren't using
frank has quit IRC. (Quit)
<knightmare> ...
```

Feature	Value
▶ sofa	[Sofa]
begin	86
end	92
PosValue	NN

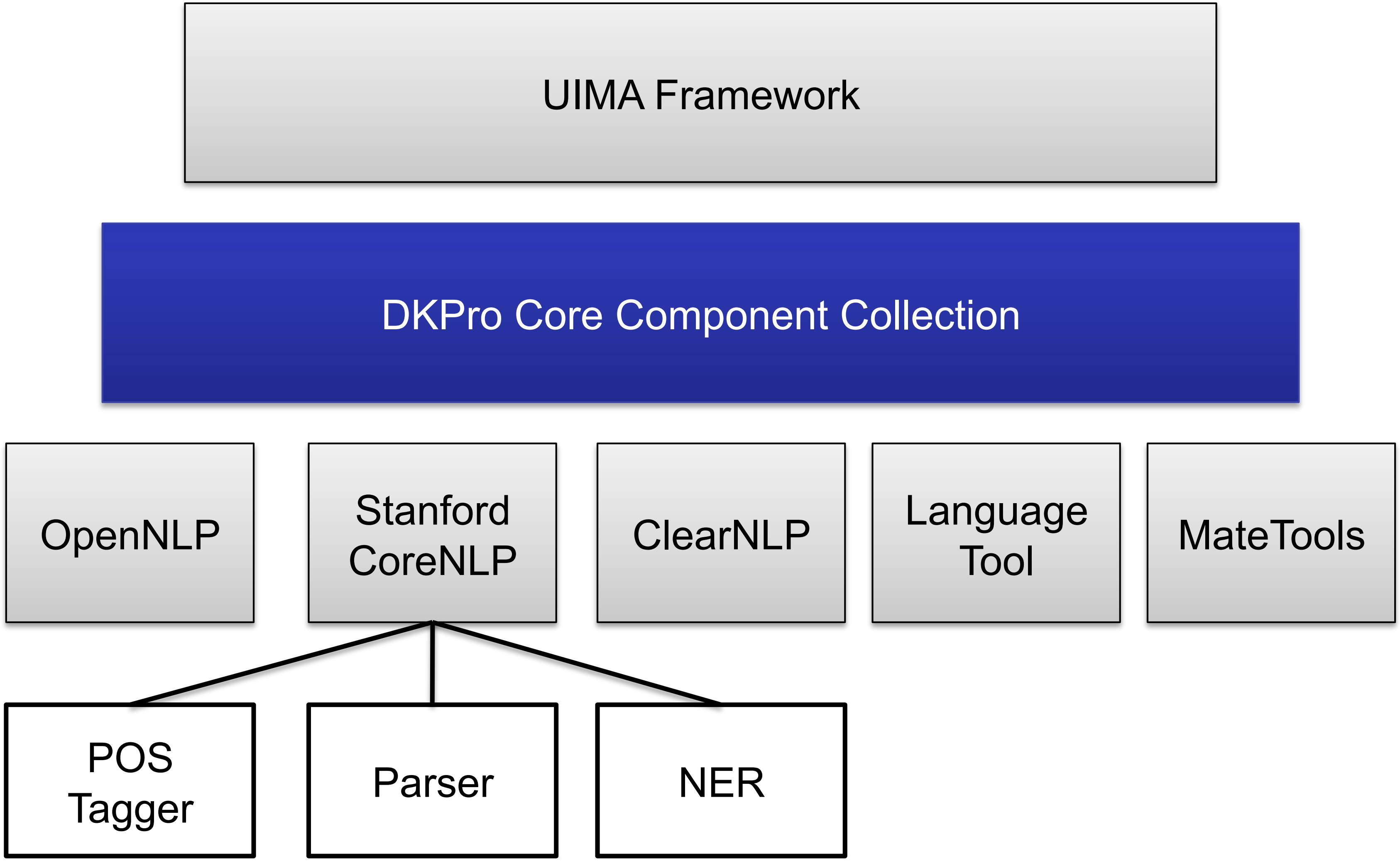
Feature	Value
▶ sofa	[Sofa]
begin	86
end	92
PosValue	NN

	Type	Style
□	Person	BACKG
□	Sentence	BACKG
□	Time	BACKG
□	Token	BACKG
☑	ADJ	BACKG
☑	ADV	BACKG
☑	ART	BACKG
☑	CARD	BACKG
☑	CONJ	BACKG
☑	N	BACKG
☑	NN	BACKG
☑	NP	BACKG
☑	O	BACKG
□	Animal	BACKG
□	Cardinal	BACKG
□	ContactInfo	BACKG
□	Date	BACKG
□	Disease	BACKG

```
Text
  CONJ
▼ ART
     all
▼ ADJ
     frank
     first
     frank
  ADV
▶ V
  N
▼ O
     <
     >
     <
     >
     ,
     (
     )
     <
     >
  CARD
▼ NN
     knightmare
     programs
     aren't
     knightmare
▼ NP
     GTA3
     IRC
  Annotation
```

DKPro Core Component Collection

What’s a component collection?



DKPro Core

- DKPro Core is an integration framework
  - Processing: tools and models
  - Primary data: corpora
  - Auxiliary data: other language resources (e.g. lexical resources)
- Primarily integration of existing work, not original work
- Contribution of DKPro Core is the integration itself
- Open Source under Apache Software License & GNU Public License

http://dkpro-core-asl.googlecode.com

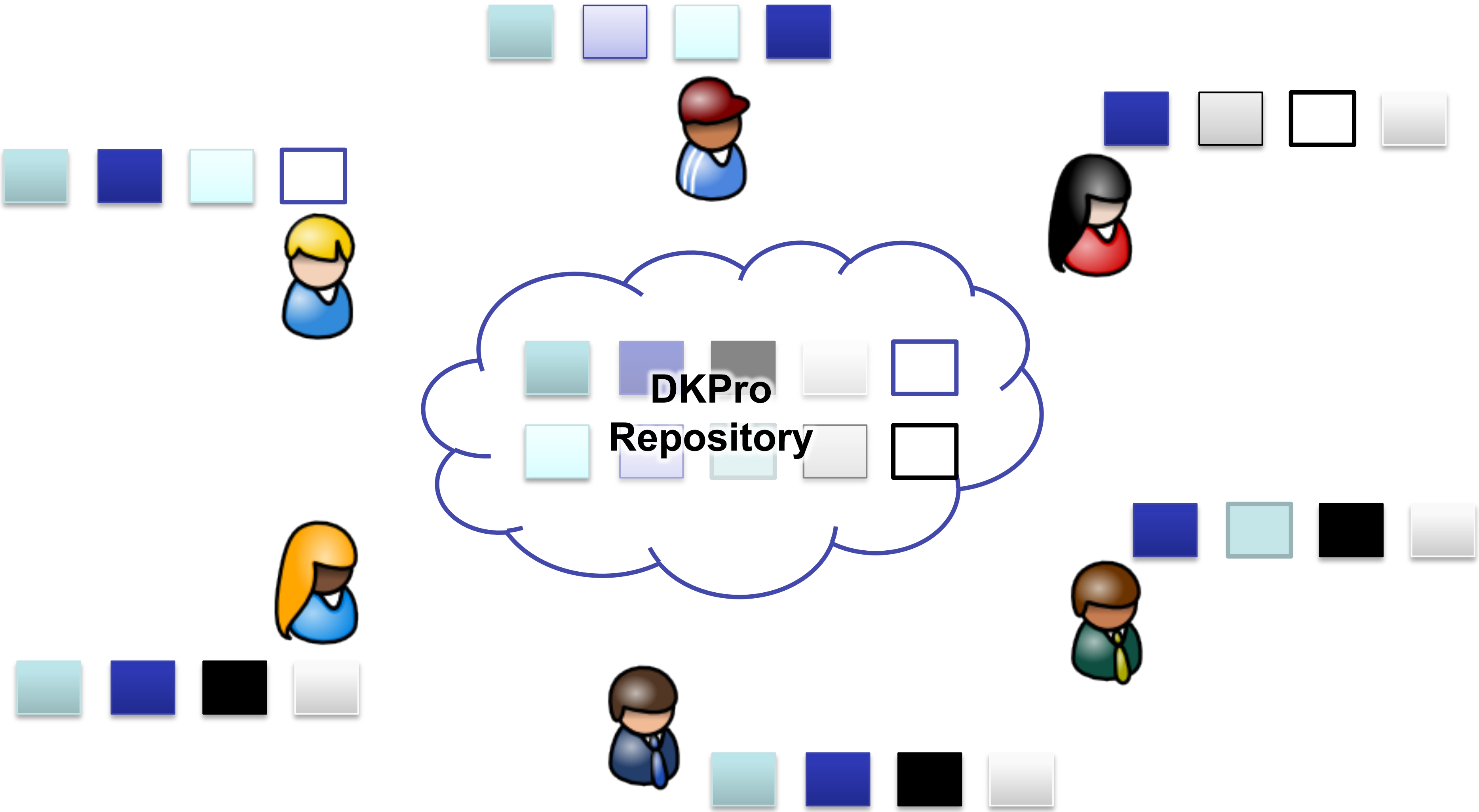
http://dkpro-core-gpl.googlecode.com

DKPro Core Philosophy

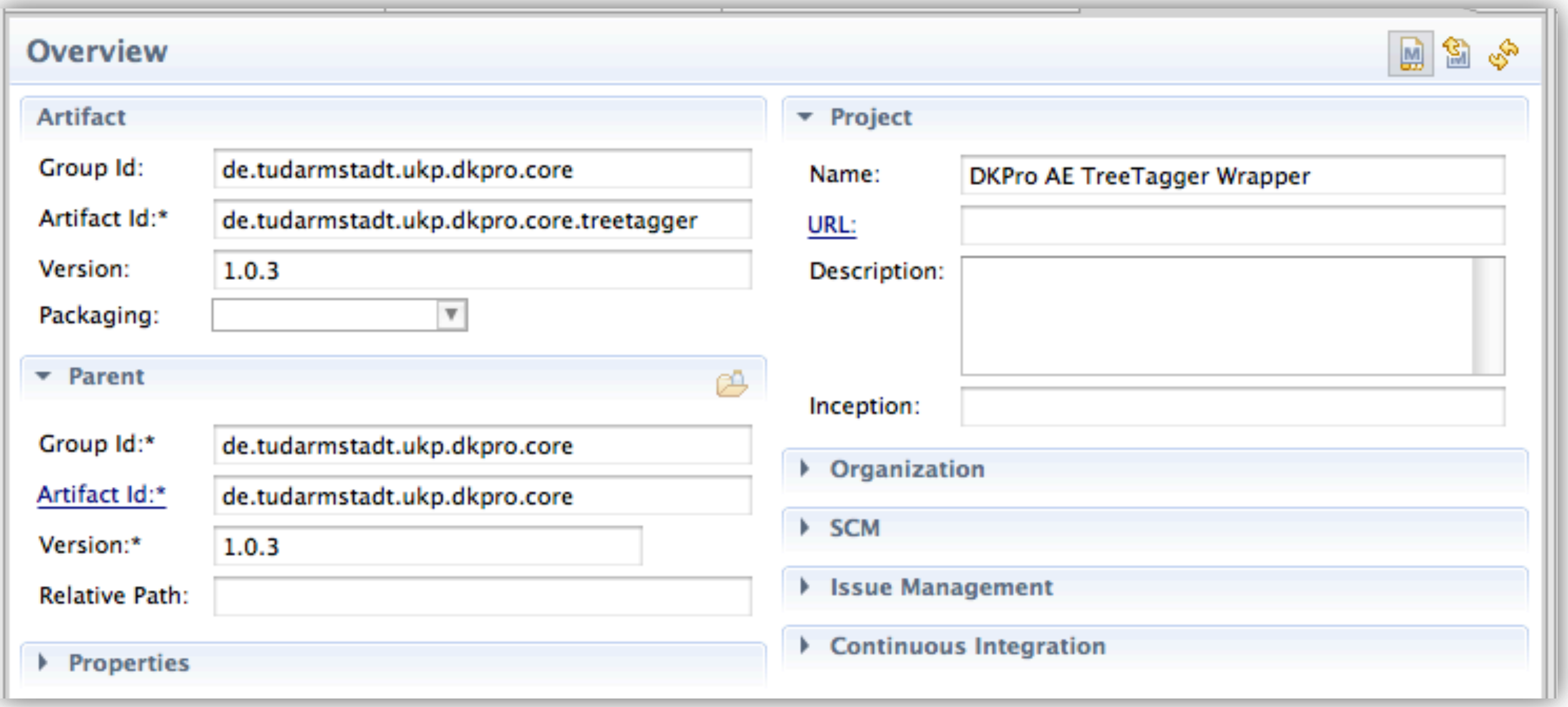
Stuff has to "just work", everywhere.

- **Simplicity**
  - Common data types used by all components
  - Common set of parameters across components
  - Sensible parameters defaults for minimal need for configuration
  - Convenient deployment of components and resources
  - Compose powerful pipelines with a few lines of code
- **Modularity**
  - Use only what you need
- **Flexibility**
  - Override parameters for fine-grained control
  - Extend data types with custom fields
  - Customize type mappings

Managing Deployment



UKP OSS Component Repository
Publish component



Overview

Artifact

Group Id:	de.tudarmstadt.ukp.dkpro.core
Artifact Id:*	de.tudarmstadt.ukp.dkpro.core.treetagger
Version:	1.0.3
Packaging:	

Parent

Group Id:*	de.tudarmstadt.ukp.dkpro.core
Artifact Id:*	de.tudarmstadt.ukp.dkpro.core
Version:*	1.0.3
Relative Path:	

Properties

Project

Name:	DKPro AE TreeTagger Wrapper
URL:	
Description:	
Inception:	

Organization

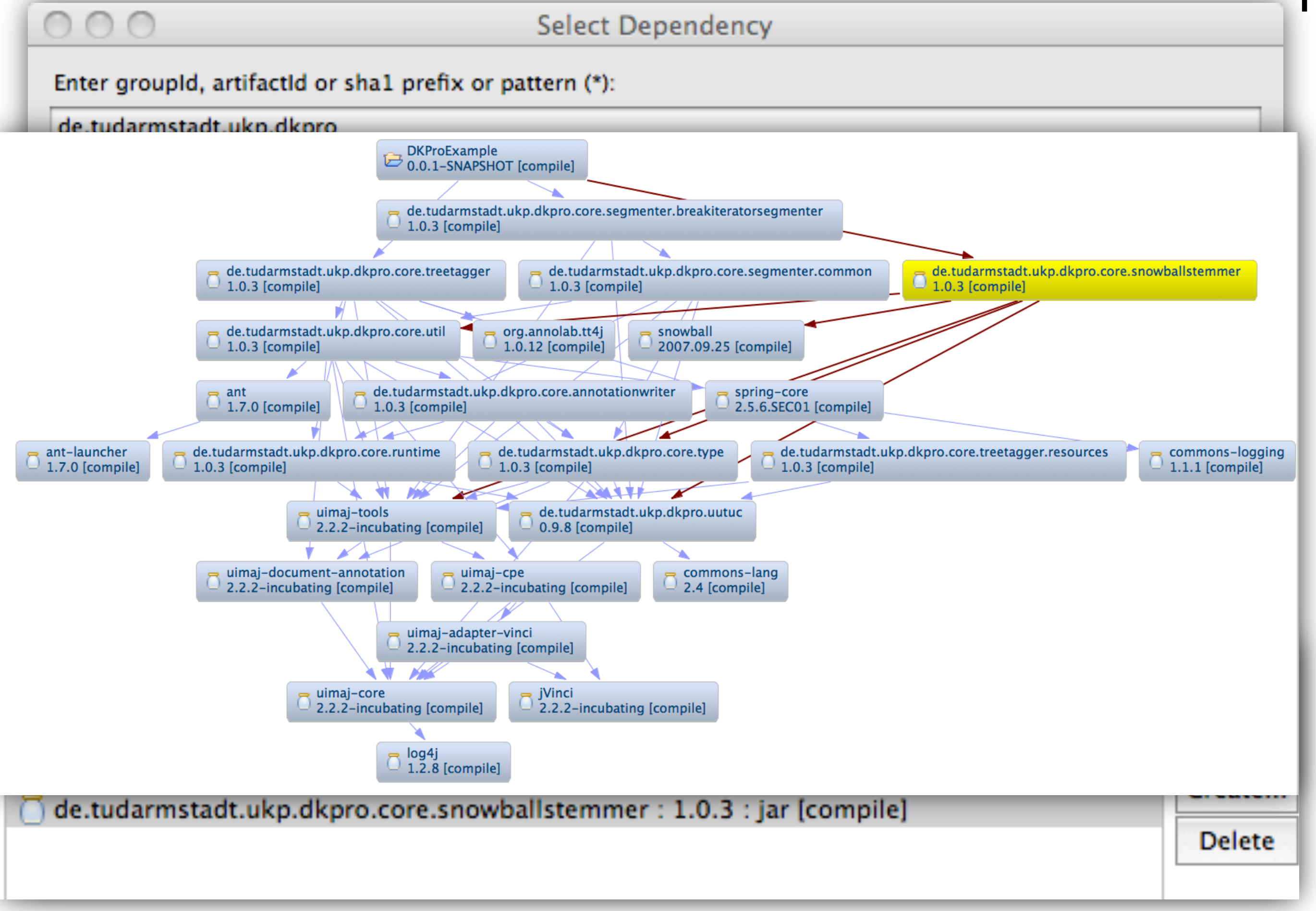
SCM

Issue Management

Continuous Integration

UKP OSS Component Repository
Retrieving components

Component Repository



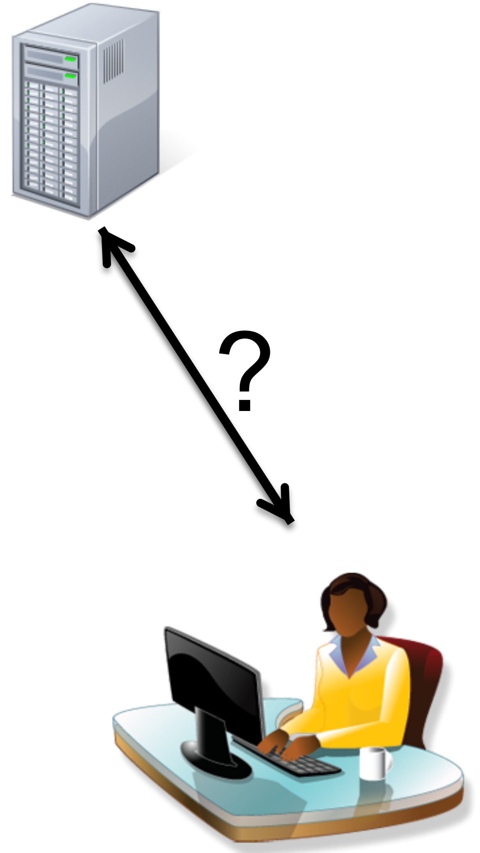
Select Dependency

Enter groupId, artifactId or sha1 prefix or pattern (*):

de.tudarmstadt.ukp.dkpro

de.tudarmstadt.ukp.dkpro.core.snowballstemmer : 1.0.3 : jar [compile]

DKProExample 0.0.1-SNAPSHOT [compile]
de.tudarmstadt.ukp.dkpro.core.segmenter.breakiteratorsegmenter 1.0.3 [compile]
de.tudarmstadt.ukp.dkpro.core.treetagger 1.0.3 [compile]
de.tudarmstadt.ukp.dkpro.core.segmenter.common 1.0.3 [compile]
de.tudarmstadt.ukp.dkpro.core.snowballstemmer 1.0.3 [compile]
de.tudarmstadt.ukp.dkpro.core.util 1.0.3 [compile]
org.annolab.tt4j 1.0.12 [compile]
snowball 2007.09.25 [compile]
ant 1.7.0 [compile]
de.tudarmstadt.ukp.dkpro.core.annotationwriter 1.0.3 [compile]
spring-core 2.5.6.SEC01 [compile]
ant-launcher 1.7.0 [compile]
de.tudarmstadt.ukp.dkpro.core.runtime 1.0.3 [compile]
de.tudarmstadt.ukp.dkpro.core.type 1.0.3 [compile]
de.tudarmstadt.ukp.dkpro.core.treetagger.resources 1.0.3 [compile]
commons-logging 1.1.1 [compile]
uimaj-tools 2.2.2-incubating [compile]
de.tudarmstadt.ukp.dkpro.uutuc 0.9.8 [compile]
uimaj-document-annotation 2.2.2-incubating [compile]
uimaj-cpe 2.2.2-incubating [compile]
commons-lang 2.4 [compile]
uimaj-adapter-vinci 2.2.2-incubating [compile]
uimaj-core 2.2.2-incubating [compile]
jVinci 2.2.2-incubating [compile]
log4j 1.2.8 [compile]
Delete



Tools and Formats

Integrated Tools

- Stanford NLP
- OpenNLP
- Mate-Tools
- ClearNLP
- LanguageTool
- TreeTagger
- JWordSplitter
- Snowball Stemmer
- TextCat
- MaltParser
- MstParser
- BerkeleyParser
- MeCab
- Jazzy
- …

Supported Formats

- Text
- PDF
- TIGER XML
- TEI XML
- BNC XML
- Negra Export
- SQL Databases
- Google web1t n-grams
- …

Readers and Writers

- **Common parameters**
  - Source / target location
  - Source / target encoding
  - ANT-like patterns (for readers)
  - Language (for readers)

- **Common features**
  - Read data from file system, ZIP/JAR archives or classpath
  - Preserve directory structure on write for recursive reads

Some currently supported corpora / resources

▪ British National Corpus	(BNC-XML)
▪ Wacky Corpora	(OpenCWB format)
▪ TüBa D/Z	(NEGRA export format)
▪ Tiger Corpus	(Tiger XML)
▪ Digitale Bibliothek	(TEI XML)
▪ Brown Corpus	(TEI XML)
▪ ACL Anthology Reference Corpus	(Text)

- …

- Google Web1T n-grams
  - Can also easily build your own n-gram database with DKPro/jweb1t

We are working on supporting more corpora and corpus formats

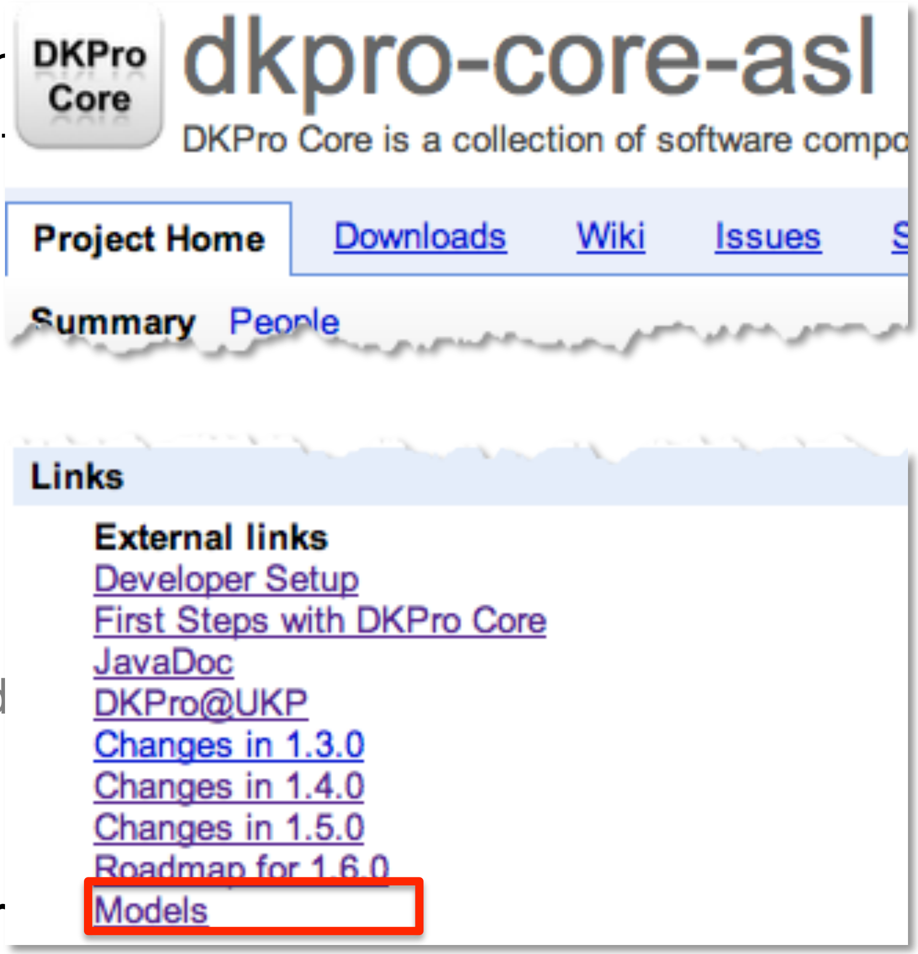
Good range of pre-trained models

- Upstream models packaged for conver
- Package includes additional model me

- 90+ models
- 20+ tools
- 15+ languages

- Best supported
  - English (Penn Treebank Tagset, Stanford
  - German (STTS Tagset, Negra/Tiger)

- Unfortunately not all models are redistr
- … we are interested in creating/collecting more models



DKPro Core

dkpro-core-asl

DKPro Core is a collection of software compo

Links

External links

- Developer Setup
- First Steps with DKPro Core
- JavaDoc
- DKPro@UKP
- Changes in 1.3.0
- Changes in 1.4.0
- Changes in 1.5.0
- Roadmap for 1.6.0
- Models

DKPro Type System Overview

Meta Data

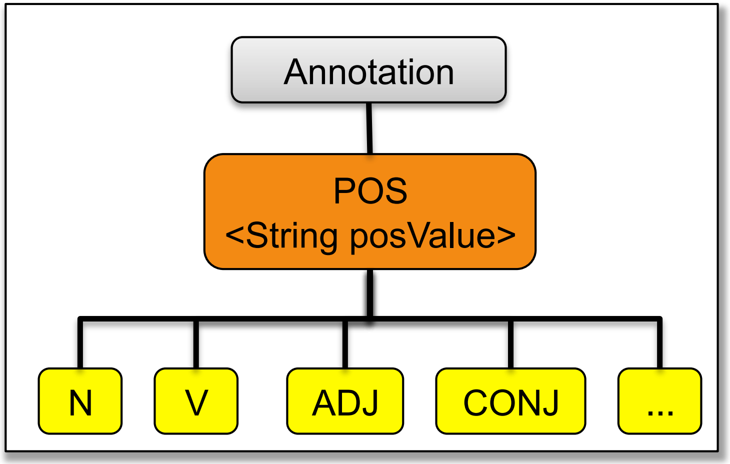
DocumentMetaData
<String documentTitle>
<String collectionId>
<String documentId>
<String documentUri>
<String documentBaseUri>

Segmentation

- Document
- Heading
- Paragraph
- Sentence
- Token
- Compound
- Split
- CompoundPart
- LinkingMorpheme
- Ngram
- StopWord

LexMorph

- POS
- Stem
- Lemma
- Morpheme



Syntax

- Constitutent
- Dependency
- Chunk

Coreference

- CoreferenceChain
- CoreferenceLink

Semantic Role Labeling

- SemanticPredicate
- SemanticArgument

Named Entities

- NamedEntity
- Location
- Person
- ...etc...

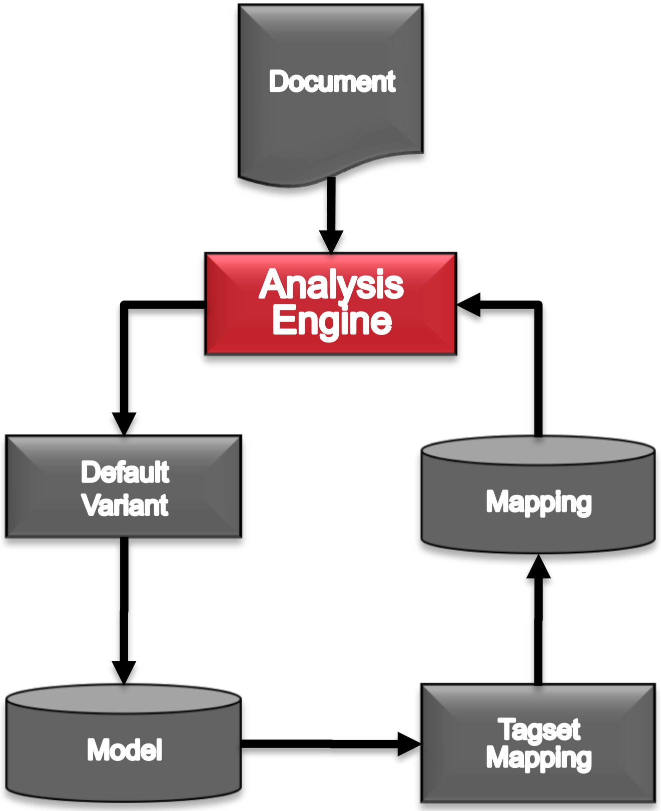
UIMA type mappings

- Tags mapped to UIMA types (configurable)
- **Generic:** Original tags stored in a *value* feature, e.g. `POS.value`
- **Coarse Grained:** Currently supported for Part-of-Speech tags
  - 13 coarse grained part-of-speech tags
  - ADJ, ADV, ART, CARD, CONJ, N (NP, NN), O, PP, PR, V, PUNC
- Convenient coarse-grained processing across languages
- Similar “Universal Part-of-Speech” tag-set published @ LREC 2012
  - *Slav Petrov, Dipanjan Das and Ryan McDonald*
  - Defines mappings for 25 tagsets in 22 languages
  - Will be adopted in a future DKPro Core release
- Similar coarse-grained mappings may come for syntax, dependencies, …

Analysis Engines

- **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



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 and 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 }
```

Hands-on

https://dl.dropboxusercontent.com/u/11205710/iuahsdfhksad/ws-tutorial.zip

Example projects

- **gscl2013-types**
  - Defining and using your own annotation types
- **gscl2013-pipeline**
  - Four ways of building a UIMA pipeline
    1. No reader, no writer
    2. No reader, with writer
    3. With reader, no writer
    4. With reader, with writer
  - Simple interactive analysis pipeline
  - Extended interactive analysis pipeline with language detection
- **gscl2013-dkpro**
  - DKPro Core monster pipeline wildly mixing components from different vendors
- **gscl2013-pos-ensemble**
  - Building an improved part-of-speech tagger using an ensemble of taggers
- **gscl-2013-ruta**
  - Running Ruta within an uimaFIT pipeline