Text Analysis and Feature Extraction in AITools 4 based on Apache UIMA

Henning Wachsmuth

August 21st, 2015

Bauhaus-Universität Weimar

www.webis.de

Outline

Covered in these slides

- Apache UIMA at a glance
- Apache UIMA components in Altools 4
- Feature extraction based on Apache UIMA
- Feature types in Altools 4
- Altools 4 – Getting started

Not Covered

- Foundations of text analysis and feature extraction (see literature)
- Details on UIMA libraries, tools, scaleout, etc. (see http://uima.apache.org)

Apache UIMA at a glance

Apache UIMA overview

Idea

- Software framework for uniform handling of text analysis
- Text analysis seen a process of annotation steps

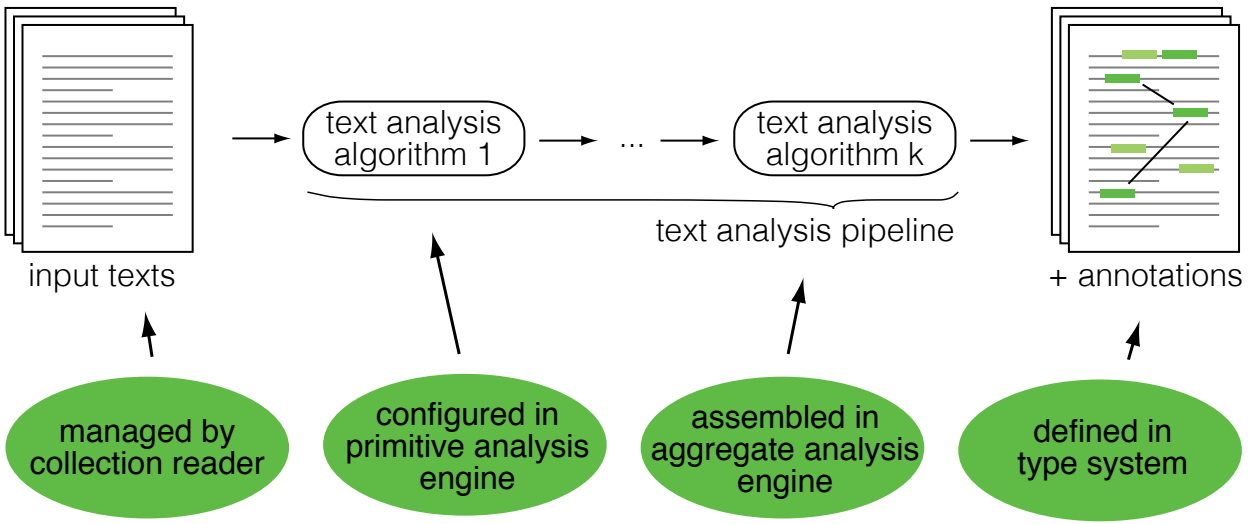
General UIMA process

- Input is (usually) plain text
- Each step adds annotations of certain types to the text
- Output is plain text + all added annotations

Main UIMA components

- Type systems
- Collection readers
- Primitive analysis engines
- Aggregrate analysis engines

General UIMA Process



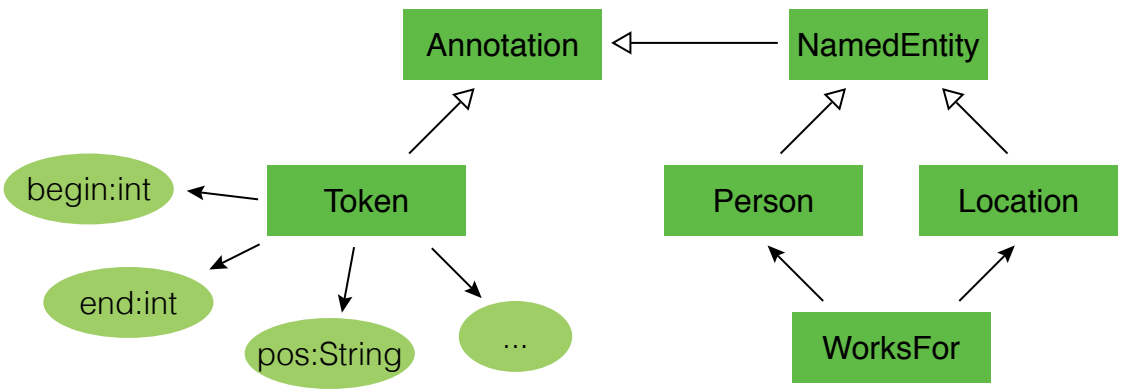
The UIMA approach

- Each component specified by an XML file
- All components work together in the same way
- Often no need to care about implementation

Main UIMA components: Type system

Annotation

- A span of text with a defined type and possible attributes
- Defined by start and end index and pointer to text
- Also used (in UIMA) to specify relations between other annotations



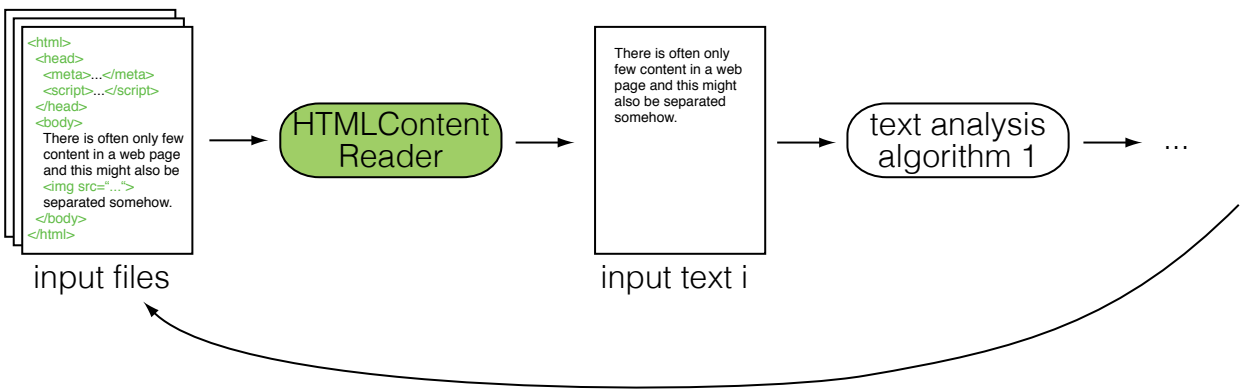
Type system

- Defines all known annotation types and their attributes
- Can be organized hierarchically
- Usually specified only once in a project
- Extension easy, modification problematic

Main UIMA components: Collection reader

Input texts

- UIMA aims at collections of input texts
- Usually one file type per collection (txt, html, ...)
- Text analysis mostly works on plain text



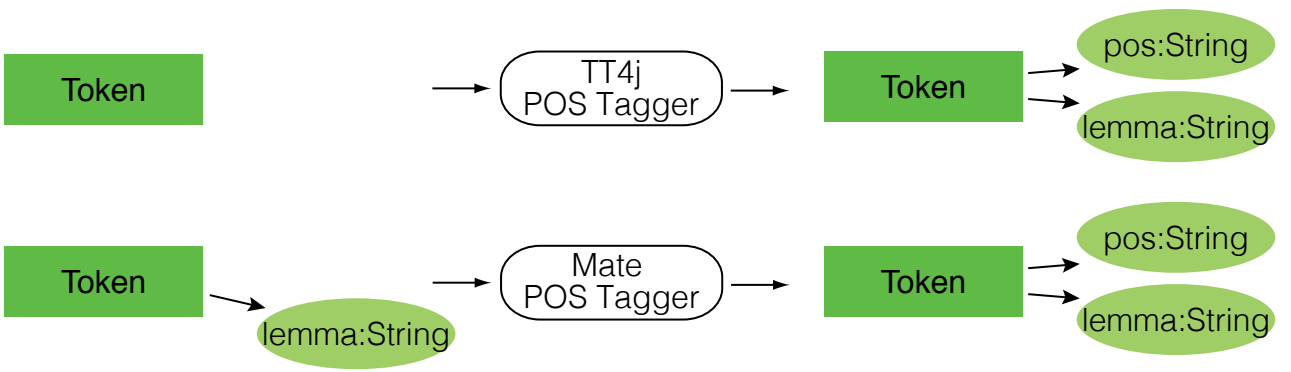
Collection reader

- Iterates over a collection of files
- Transforms each file into plain text (content)
- Usually one collection reader per file type
- Specified once and then reused

Main UIMA components: Primitive analysis engines

Text analysis algorithm

- Usually performs one text analysis
- Requires certain input types and produces certain output types
- Some generally usable, others specific to language, domain, application, ...
- Different algorithms for the same analysis exist



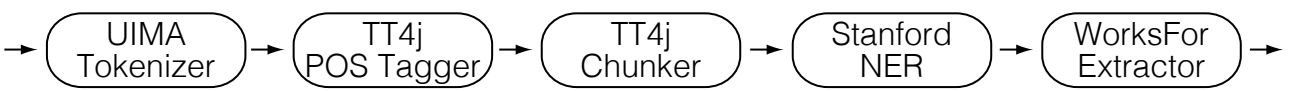
Primitive analysis engine

- Defines one configuration of one text analysis algorithm
- Many algorithms have only one, some have several
- Configuration can also be set from application

Main UIMA components: Aggregate analysis engines

Text analysis pipeline

- A sequence of text analysis algorithms
- Realizes a complete process to produce certain output types
- Input requirements of all algorithms must be met



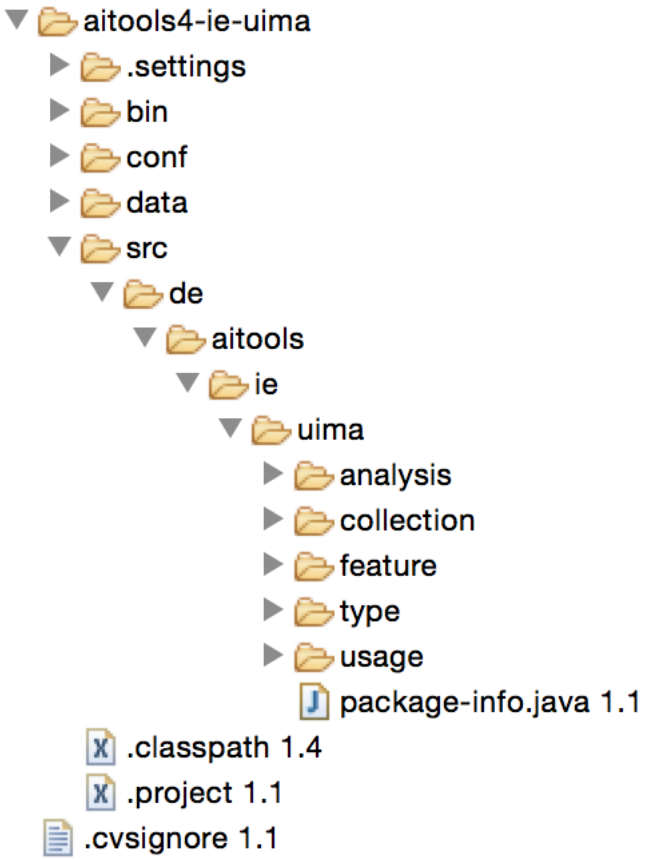
Aggregate analysis engine

- Specifies a pipeline of primitive analysis engines
- No actual implementation
- Often application-specific

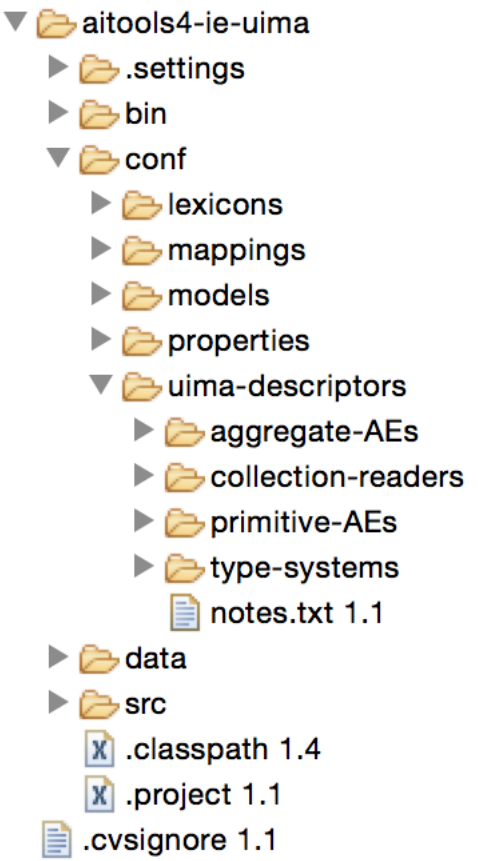
Apache UIMA components in Altools 4

Altools 4: UIMA overview

Source code in CVS



Components in CVS



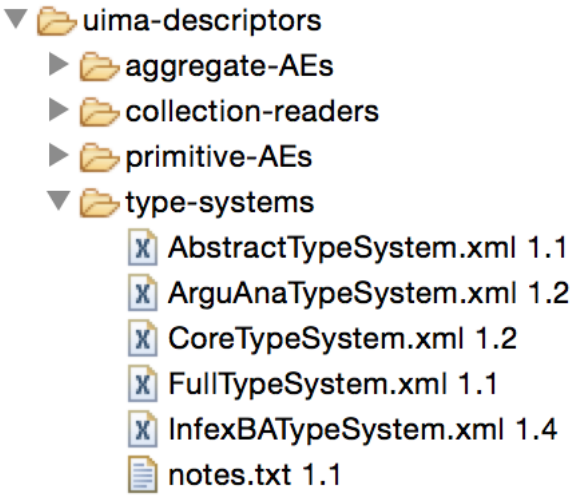
Altools 4: Realized type systems

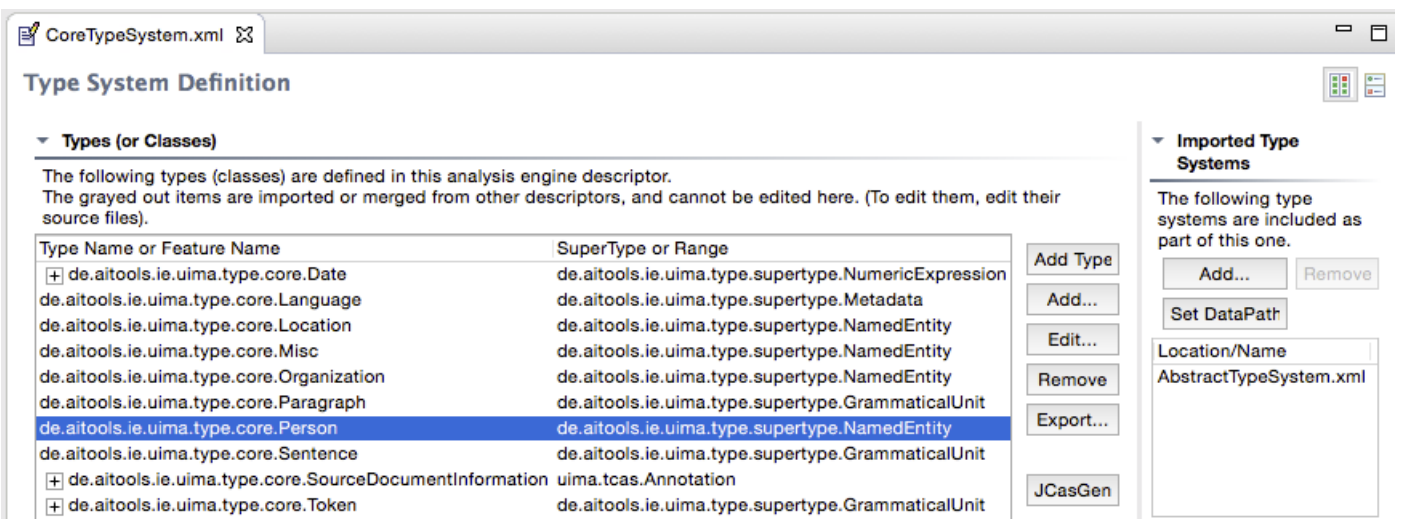
Current state

- Abstract and core types for general use
- Specific types from ArguAna and InfexBA

Our organization

- New type systems build on existing type systems

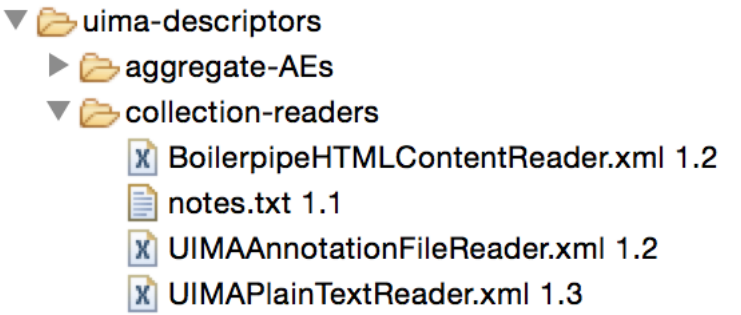


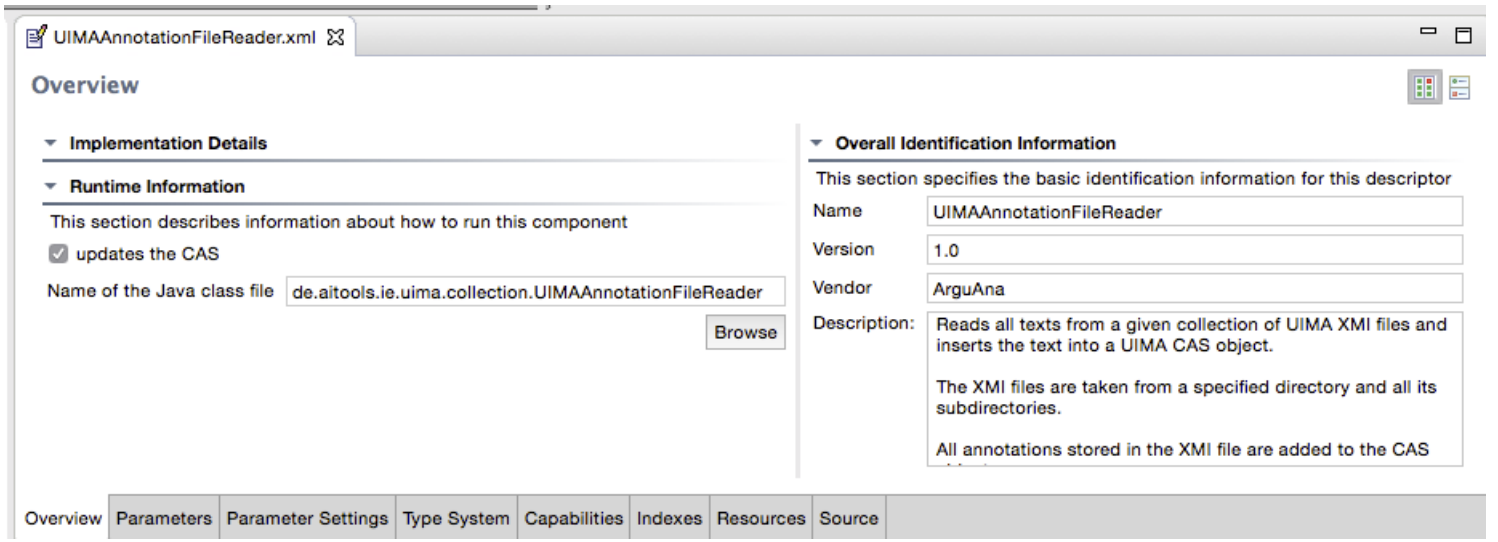


Altools 4: Realized collection readers

Current state

- Readers only for most common file types
- Some more to come, e.g., for input files with one token per line
- Others can be implemented quickly when needed

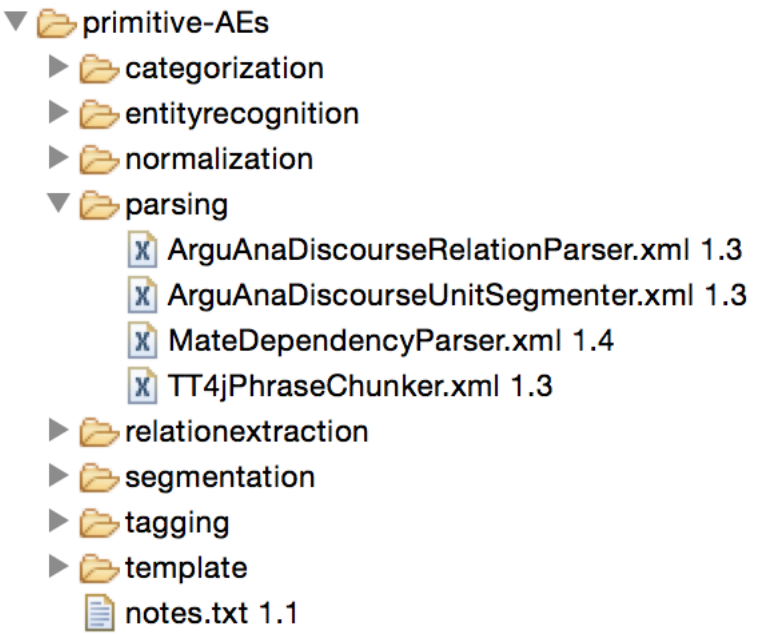




Altools 4: Realized primitive analysis engines

Current state

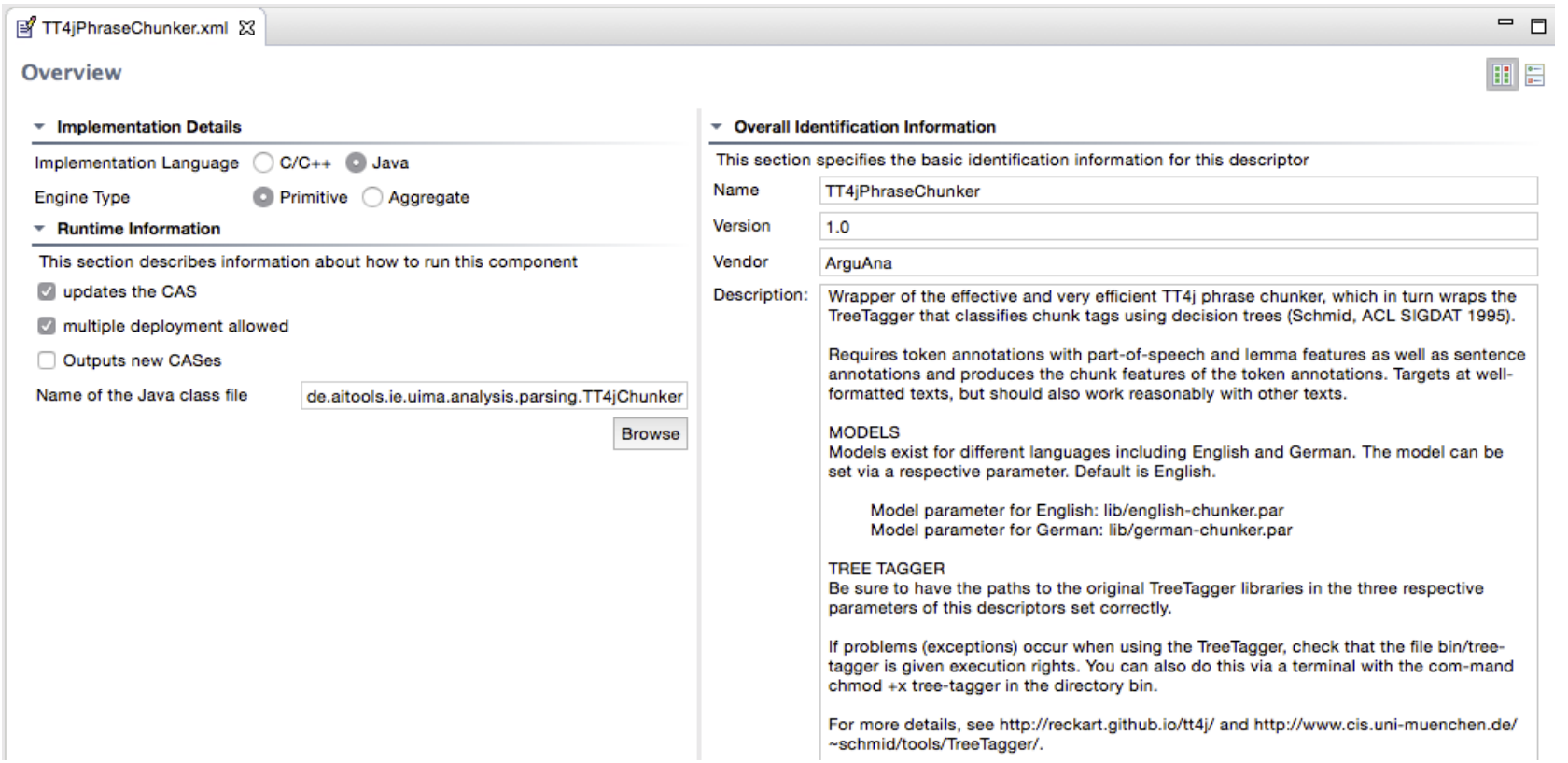
- 36 text analysis algorithms represented
- Most of them for English and/or German
- About 20 of them should be reusable for several purposes
- Rest specific for tasks like those from InfexBA and ArguAna
- More to come



Our organization

- Grouped according to general type of analysis
- File names roughly in the form *<Provider><Information><Approach>*

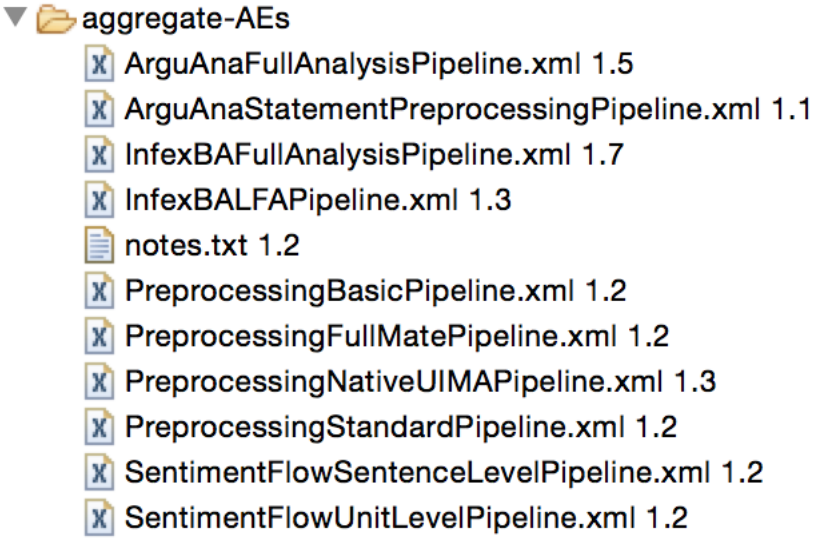
Altools 4: Realized primitive analysis engines (2)

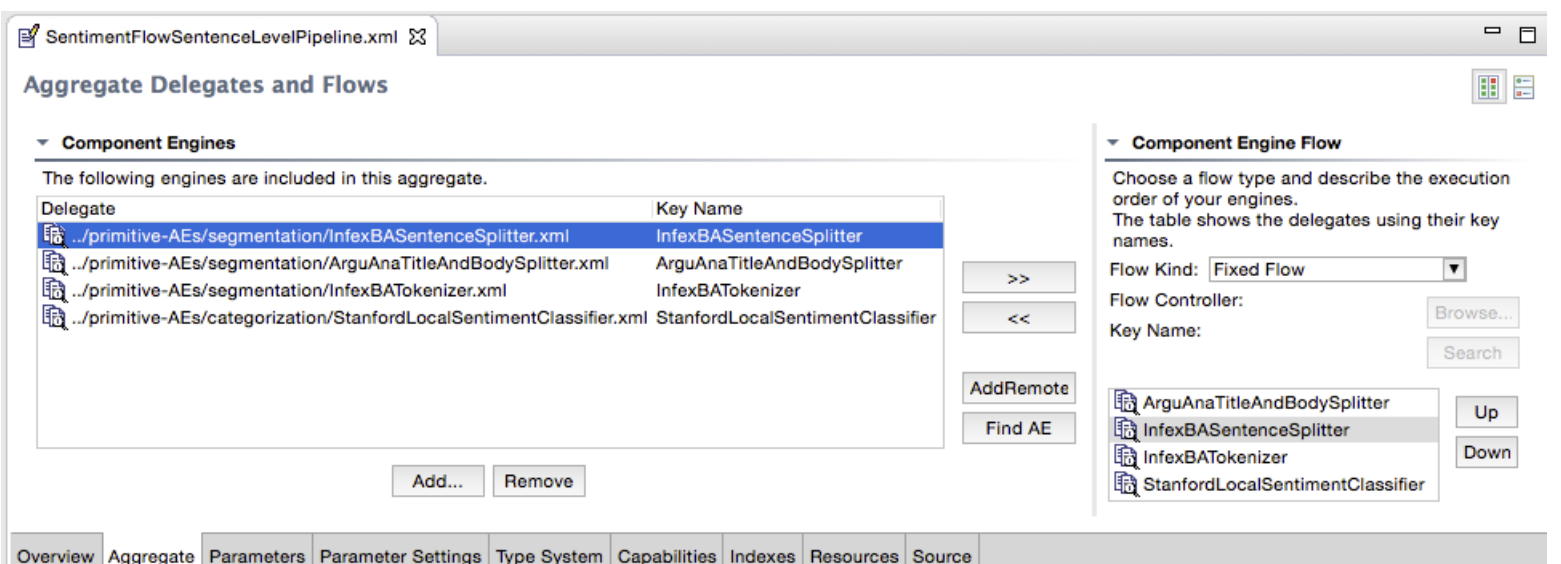


Altools 4: Realized aggregate analysis engines

Current state

- Main pipelines from InfexBA and ArguAna preserved
- Some further examples provided
- Others can be implemented quickly when needed



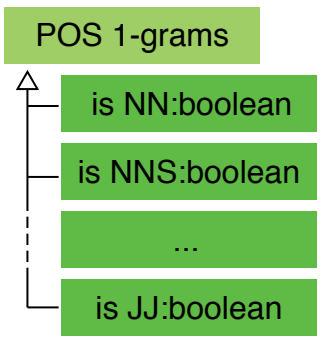


Feature extraction based on Apache UIMA

Feature extraction overview

Feature

- Any measurable property of an object, here of a text or text fragment
- One feature value per feature
- A vector of feature values defines a model of an object
- Needed for machine learning



Feature type

- A set of features of the same kind
- Vector contains feature values for a set of feature types

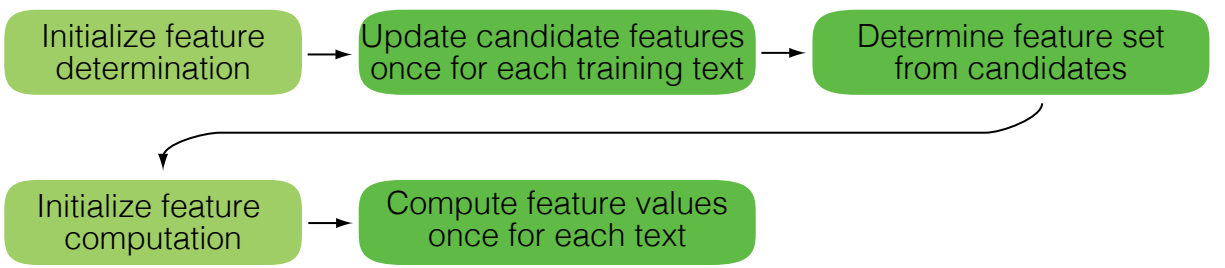
Feature extraction

- Computation of the values of all considered features of a text (fragment)
- Comes after text analysis
- Not covered by Apache UIMA itself

Our feature framework based on Apache UIMA

Feature type interface

- One (Java) interface that each feature type implements
- Highly parametrizable
- Aggregate feature types orchestrate single feature types
- 3 main methods + 2 initialization methods



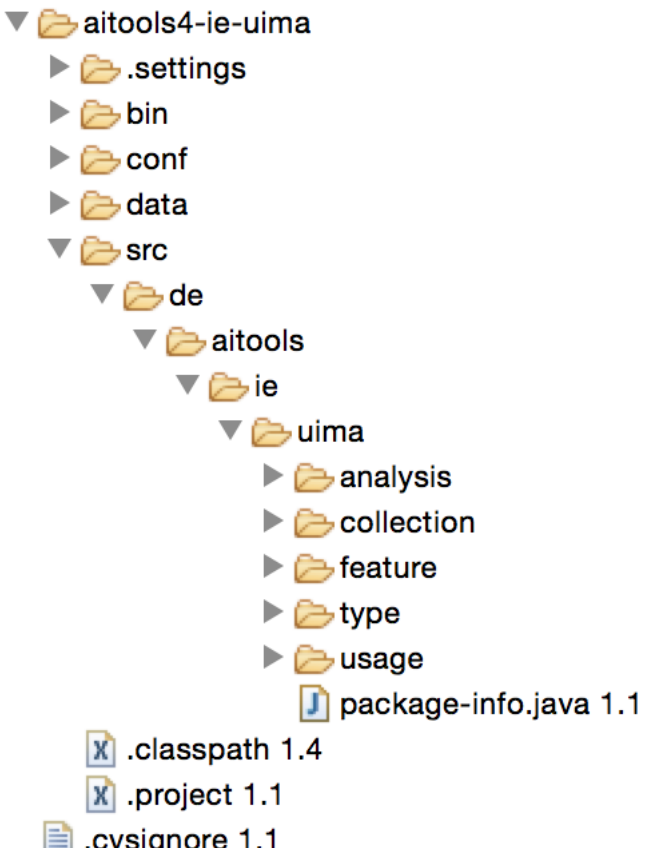
Feature framework

- Interface implies feature extraction process
- Determination of feature set on a training set (where needed)
- Computation of feature values on any text
- Can be directly integrated into UIMA text analysis process

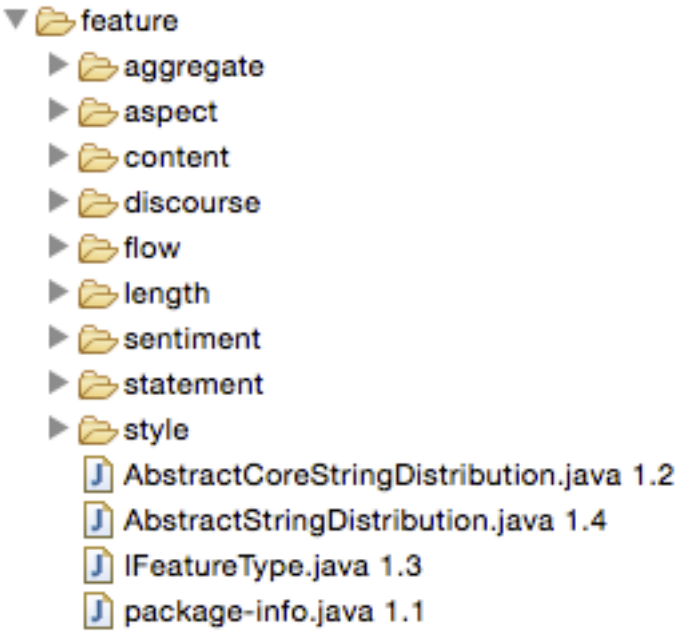
Feature types in Altools 4

Altools 4: Feature overview

Source code in CVS



Feature package within source code



Altools 4: Realized feature types

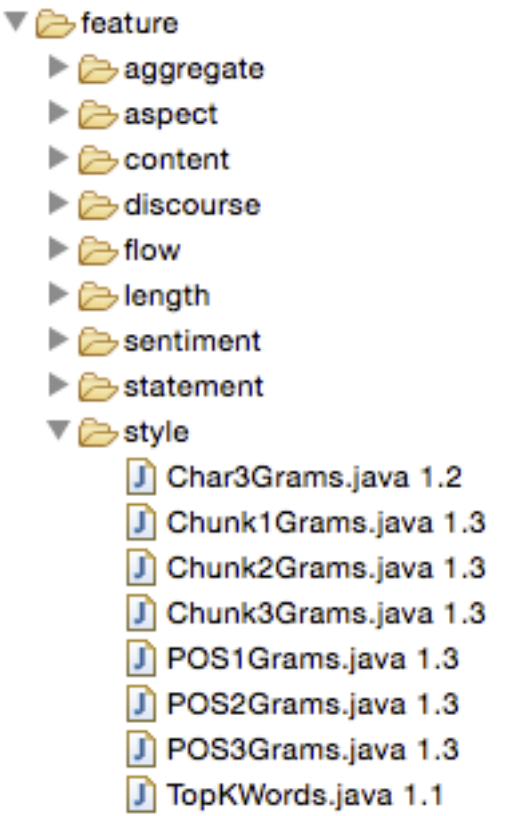
Current state

- 47 single feature types
- About 25 of them should be reusable for several purposes
- Rest specific for tasks like those from InfexBA and ArguAna
- More to come

- A handful of exemplary aggregate feature types
- Two abstract feature types for easy extension

Our organization

- Types grouped according to the kind of information that is predominantly captured



Altools 4 – Getting started

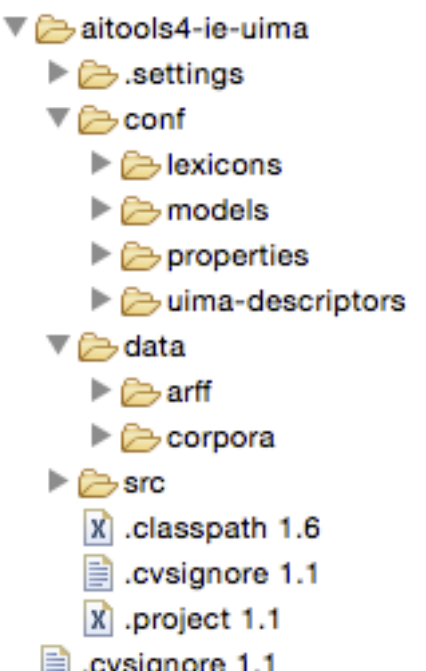
Altools 4 – Setup

Required code bases

- Folder code-in-progress/aitools/aitools4-ie-uima
- Folder thirdparty
- Both folders should be on the same level (in Eclipse) to ensure that third-party models are found

Altools 4 structure

- Folder conf: Lexicons & models of algorithms, properties of applications, descriptor files of UIMA components
- Folder data: Collections of input texts (so called corpora) and output files (such as ARFF files)
- Folder src: All provided source code for text analysis, feature extraction, etc.



Your code

- Create your own project
- Set Altools 4 as a referenced project

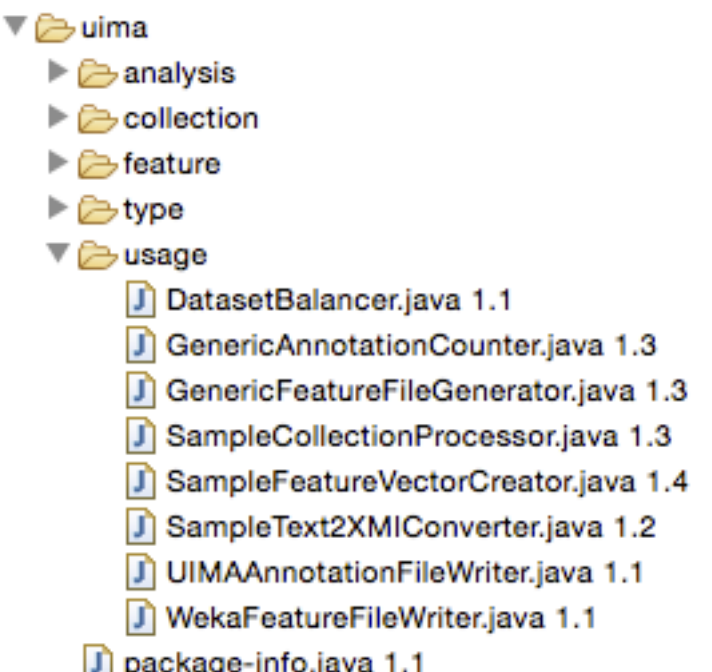
Using the existing algorithms and feature types

Coverage

- Algorithms for most standard preprocessing
- Feature types for many typical baseline approaches
- Some more advanced approaches for tasks like sentiment analysis
- All also serve as implementation examples

Applications to illustrate usage

- Sample processing of a collection of texts
- Sample conversion from text to XMI files
- Sample creation of feature vectors
- Generic counting of annotations
- Generic generation of Weka ARFF feature files



Further files

- Samples of English and German texts
- Parameter files for feature generation etc.

Developing your own UIMA text analysis algorithm

- Only two methods of general interface to be implemented usually
- Wrappers of existing libraries also implemented in this way

```
public class MyTextAnalysisAlgorithm extends JCasAnnotator_ImplBase {
    public void initialize(UimaContext aContext) throws ... {
        super.initialize(aContext);
        // Load parameters, initialize objects, etc.
    }
    public void process(JCas jcas) {
        // Process text and annotations in the given JCas data structure
    }
}
```

- Some superclasses for common text analyses already implemented

```
public class SentimentClassifier extends AbstractWekaTextClassifier {
    protected Annotation createAnnotation(JCas jcas, Annotation a,
            String classLabel){
        Sentiment s = new Sentiment(jcas, a.getBegin(), a.getEnd());
        s.setPolarity(classLabel);
        return s;
    }
}
```

Developing your own feature type

- Feature type interface specifies five methods as described above

```
public void initializeFeatureDetermination(Properties configProps);
public void updateCandidateFeatures(JCas jcas, int start, int end);
public List<String> determineFeatures(Properties configProps,
        Properties normalizationProps);
public void initializeFeatureComputation(List<String> featureNames,
        Properties configProps, Properties normalizationProps);
public List<Double> computeNormalizedFeatureValues(JCas jcas, int start,
        int end);
```

- Additionally, some superclasses for common feature types provided

```
public class POS1Grams extends AbstractStringDistribution {
    public POS1Grams(){
        super("POS1Grams", /* ... further parameters... */);
    }
    protected String getStringFromAnnotation(Annotation annotation,
            boolean caseSensitive, boolean isLast){
        return ((Token) annotation).getPos();
    }
}
```

Getting started – recommended order

Explore source code

1. Some applications in the usage package, begin with the simple ones
2. Some text analysis algorithms, begin with the wrappers of external libraries
3. The feature type interface `IFeatureType`
4. Some feature types, begin with `AbstractStringDistribution`
5. ...

Explore UIMA components

1. Some primitive analysis engines, begin with those of the explored algorithms
2. One or two aggregate analysis engines
3. Some type systems, begin with `CoreTypeSystem.xml`
4. ...

Explore Apache UIMA

1. Read manuals at http://uima.apache.org, begin with “Overview and Setup”
2. Start developing some UIMA-based applications

Final remarks

Source code

- Tested in applications, but certainly not free of bugs
- Thorough Javadoc comments, but may partly be incomplete or outdated

  $\rightarrow$ *If you find something, let me know*

Usage

- Some parts of the frameworks need to be internalized first
- Some parts might reflect my view of thinking merely

  $\rightarrow$ *If you get stuck, feel free to ask me*

Apache UIMA

- De facto standard, well-designed, many advantages
- Some annoying details do exist

  $\rightarrow$ *If you cannot solve some problem, I might be able to help*