ISMLA Session 5 - UIMA

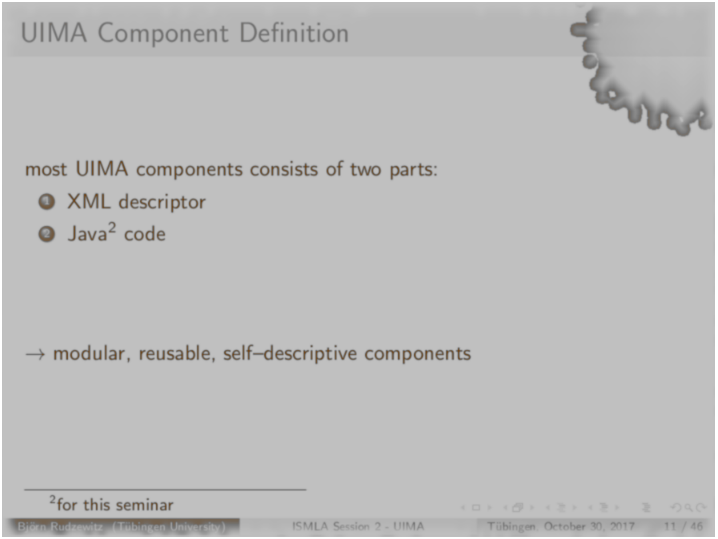
Björn Rudzewitz

Tübingen University

November 20, 2017

Plan

Flashback: First UIMA Session



UIMA Component Definition

most UIMA components consists of two parts:

1. XML descriptor
2. Java[2] code

→ modular, reusable, self–descriptive components

[2]for this seminar

Björn Rudzewitz (Tübingen University) ISMLA Session 2 - UIMA Tübingen, October 30, 2017 11 / 46

Motivation

advantages:

- explicit standoff description of Java code for reusable, documented components
- non-developers can develop pipelines/AAEs by only referring to the descriptor file
- clear division of labour: component description vs. application logic
- separate GUIs for user-friendly assembly of components

Motivation

disadvantages:

- synchronization of code and descriptor necessary
- certain information encoded twice, e.g. configuration parameter name
- descriptors needed for every variant of a parameter, e.g. testing numerical values in an interval
- refactoring code potentially difficult
- at least two files for one component doubles project size

Approach

- uimaFIT approach: only write Java code, no descriptors
- XML descriptor files and external tools (e.g. DocumentAnalyzer) replaced by programmatic solutions
- ability to dynamically work with components without (static) descriptor files

Approach

the following information is based on the developer documentation of uimaFIT[1]

two main design goals of uimaFIT:

1. simpler implementation of components
2. simpler instantiation/running/testing of components

essentially the information from XML descriptors and external tools is expressed in Java objects

[1] https://uima.apache.org/d/uimafit-current/tools.uimafit.book.html

Integration of uimaFIT

add the following dependency to your pom.xml:

artifact id	uimafit-core
group id	org.apache.uima
version	2.3.0

uimaFIT tools

Different scenarios:

1. porting existing code to uimaFIT
2. simplifying UIMA code with utilities
3. implementing new components directly in uimaFIT

Porting an AE to uimaFIT

Step 1: parent class

change the annotator's parent class from

`org.apache.uima.analysis_component.JCasAnnotator_ImplBase`

to

`org.apache.uima.fit.component.JCasAnnotator_ImplBase`

Porting an AE to uimaFIT

Step 2: configuration parameters

change the config param initialization from

```
(String)
aContext.getConfigParameterValue("modelLocation");
```

to

```
public static final String PARAM_MODEL_LOCATION =
"modelLocation";
@ConfigurationParameter(name = PARAM_MODEL_LOCATION,
defaultValue="en-sent.bin",
description="the path to the sentence detector file")
private String modelLocation;
```

Porting an AE to uimaFIT

configuration parameters

- configuration parameters are assumed to be mandatory unless a `mandatory=false` field in the annotation is present
- values are automatically cast and injected into the non-final fields
- by convention every final declaration of a parameter name starts with `PARAM_`

Porting an AE to uimaFIT

Step 3: code simplification

check whether the code can be simplified via JCasUtils (see sub section 5)

Demonstration

Porting the sentence detector wrapper to uimaFIT

Running a uimaFIT pipeline

```
// create type system
TypeSystemDescription tsd = TypeSystemDescriptionFactory
.createTypeSystemDescription("src/main/resources/TypeSystem")

// create analysis engine
AnalysisEngine ae = AnalysisEngineFactory.createEngine(
SentenceDetectorFit.class, tsd,
SentenceDetectorFit.PARAM_MODEL_LOCATION, "en-sent.bin");

// create JCas for testing
JCas jcas = JCasFactory.createJCas(tsd);
jcas.setDocumentLanguage("en");
jcas.setDocumentText(input);

// run analysis engine
ae.process(jcas);
```

Views

- JCas can have different "views" / "Sofas"[2], i.e. perspectives on a document
- Example:
  - parallel corpus
  - same content expressed in different languages
  - one view for each language variant of a specific text
- Example:
  - content in mutliple modalities
  - one view for video, one for text

[2]Subject of Analysis

Views

- every JCas defines default view `_InitialView`
- attention: the `process` method of AEs when not specified differently always operates on `_InitialView`

view reference: https://uima.apache.org/d/uimaj-2.4.0/tutorials_and_users_guides.html#ugr.tug.aas

Views

Defining views in Collection Reader:

```
getNext(JCas jcas){
        JCas targetAnswer = jcas.createView("targetAnswer");
        targetAnswer.setDocumentLanguage(langCode);
        targetAnswer.setDocumentText(...)

        JCas studentAnswer = jcas.createView("studentAnswer")
        ...
...
```

Views

Processing different views:

```
...
String sofas[] = { "targetAnswer", "studentAnswer" };
AggregateBuilder builder = new AggregateBuilder();

for (String sofa : sofas) {

       AnalysisEngineDescription tokenizer =
       createEngineDescription(Tokenizer.class);
       builder.add(tokenizer, "_InitialView", sofa);
...

}
AnalysisEngine sofaAggregate = builder.createAggregate();
...
SimplePipeline.runPipeline(reader, sofaAggregate, consumer);
```

Porting a Collection Reader

1. parent class:
   `org.apache.uima.fit.component.CasCollectionReader_ImplBase`
2. `initialize` with additional argument UIMAContext
3. config parameters like for AE

Porting a CAS Consumer

1. parent class:
   `org.apache.uima.fit.component.JCasAnnotator_ImplBase`
   i.e. consumers are AEs with uimaFIT !
2. cf. AE port instructions

SimplePipeline

```
TypeSystemDescription tsd = TypeSystemDescriptionFactory
.createTypeSystemDescription("src/main/resources/TypeSystem")

CollectionReaderDescription reader =
CollectionReaderFactory.createReaderDescription(
RecursiveFileReaderFit.class, tsd, RecursiveFileReaderFit.PARA
RecursiveFileReaderFit.PARAM_LANG, "en");

AnalysisEngineDescription sent =
AnalysisEngineFactory.createEngineDescription(SentenceDetecto
SentenceDetectorFit.PARAM_MODEL_LOCATION, "en-sent.bin");

AnalysisEngineDescription consumer = AnalysisEngineFactory.cre
TableConsumerFit.PARAM_OUTPUT_DIR, outputDir);

SimplePipeline.runPipeline(reader, sent, consumer);
```

JCasUtil

select

replace

```
Iterator sentIter = arg0.getAnnotationIndex(Sentence.type)
.iterator();
while (sentIter.hasNext()) {
      Sentence sent = (Sentence) sentIter.next();
```

by

```
for (Sentence sent : JCasUtil.select(arg0, Sentence.class)) {
```

JCasUtil

selectCovered

replace

```
Iterator tokenIter = arg0.getAnnotationIndex(Token.type).itera
while (tokenIter.hasNext()) {
       Token token = (Token) tokenIter.next();
       if (token.getBegin() >= sent.getBegin()
           && token.getEnd() <= sent.getEnd()) {
```

by

```
for (Token token : JCasUtil.selectCovered(Token.class, sent))
```

JCasUtil

Utility methods

method	fetches
select(cas, type)	all annotations of this type
selectAll(cas)	all annotations
selectCovered(type, annotation)	all annotations “below” another annotation
selectPreceding(type, annotation, n)	maximally n preceding annotations of this type

full list: `https://uima.apache.org/d/uimafit-current/tools.uimafit.book.html#ugr.tools.uimafit.casutil`

Exercises

- exercies for porting UIMA components to uimaFIT and assembling pipelines
- see handout

References

parts of the slides are based on the official documentation found under
https://uima.apache.org/uimafit.html
and
https://uima.apache.org/d/uimafit-current/tools.uimafit.book.html
(last accessed 2017-11-14)

the official uimaFIT article is [Ogren and Bethard, 2009]

Philip Ogren and Steven Bethard. Building test suites for UIMA components. In *Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing (SETQA-NLP 2009)*, pages 1–4, Boulder, Colorado, June 2009. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W/W09/W09-1501`.