ISMLA Session 2 - UIMA

Björn Rudzewitz

Tübingen University

October 30, 2017

UIMA

“Market analyses indcating a growing need to process unstructured information, specifically multilingual, natural text [. . .] led to the development of [. . .] UIMA.”

- Ferrucci and Lally [2004]

UIMA

"UIMA is an open, industrial-strength, scaleable and extensible platform for creating, integrating and deploying unstructured information management solutions from powerful text or multi-modal analysis and search components."

- Official Documentation[1]

[1] https://uima.apache.org/d/uimaj-current/overview_and_setup.html#ugr.ovv.conceptual

Plan

Introduction

- UIMA = Unstructured Information Management Architecture
- goal: bring structure to unstructured information
- in computational linguistics: bring structure to texts/extract information

references: Ferrucci and Lally [2004], Gotz and Suhre [2004], `https://uima.apache.org/` (last accessed 2017-10-11)

Introduction

UIMA

Introduction

$$\text{Unstructured Information} \quad \overset{\text{UIMA}}{\Rightarrow} \quad \text{Structured Information}$$

Introduction

Unstructured Information	$\overset{\text{UIMA}}{\Rightarrow}$	Structured Information
loosely/undefined semantics		well–defined, explicit semantics
e.g. text, audio, . . .		e.g. indexed keywords, data base entries, ML tables, . . .

Feedback Loop

- last session: assigning classes and finding patterns not necessarily independent
- "feedback loop" [Ferrucci and Lally, 2004]:
    - from unstructured to structured information
    - use structured information to inform the structuring tools to struture unstructured information

Introduction

UIMA purpose/tasks:

- *"software architecture for developing UIM applications"* [Ferrucci and Lally, 2004, p. 329]
- process and annotate data, output results
- provide reusable, modular, exchangeable components
- incrementally enrich data by adding analysis layers *without* changing original data
- add annotations/analyses that (can) build on previous annotations

Introduction

- UIMA is a framework for writing analysis **pipelines**
- pipeline: process an observation (in UIMA "document") step by step and build upon previous anaylses
- pipeline initialized only once, then every observation piped through initialized components

UIMA application logic

- UIMA is a **framework**:
  application flow is defined by the framework
- task of the developer: write components for the framework according to framework's specification/design patterns
- framework "knows" how to use the components

UIMA Component Definition

most UIMA components consists of two parts:

1. XML descriptor
2. Java[2] code

$\rightarrow$ modular, reusable, self–descriptive components

[2]for this seminar

UIMA Component Definition

most UIMA components consists of two parts:

1. XML descriptor:
   what the component needs and produces ("capabilities"), required resources ("configuration parameters")
2. (Java) code
   programming logic, **how** it does what the descriptor describes

certain meta descriptors (e.g. combination of components) are only XML

UIMA Workflow

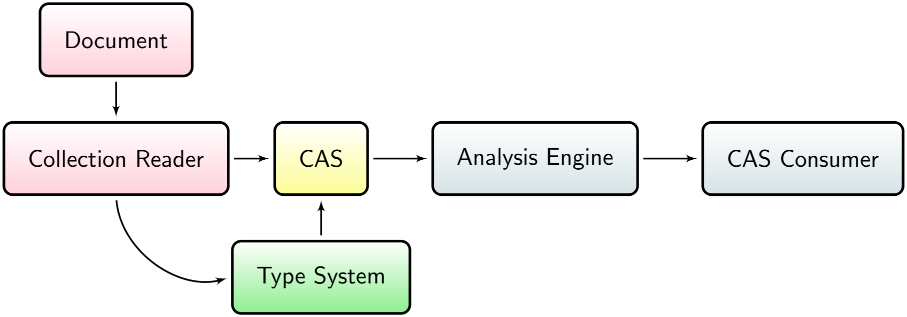

Figure: UIMA Workflow

Project Setup

Setting up environment:

- Eclipse[3]
- m2e Eclipse plugin
- UIMA Eclipse plugin:
  `http://www.apache.org/dist/uima/eclipse-update-site/`

[3] default theme

Project Setup

Setting up project:

1. Maven Java project
   File → *New* → *Other* → *Maven project*
2. add src/main/resources directory, put in build path
3. add dependencies to pom.xml

Project Setup

Dependencies (minimal)

group id	artifact id	version
org.apache.uima	uimaj-core	2.10.1
org.apache.uima	uimaj-tools	2.10.1
org.apache.uima	uimaj-document-annotation	2.10.1
org.apache.opennlp	opennlp-tools	1.8.2

Demonstration

project setup

Type System

Type System

- declares all types of annotations used in a specific application, and their relationships (single inheritance supported)
- annotation:
    - name of annotation
    - type of annotation
    - begin index
    - end index
    - (optional): additional features, each feature with a range type (String, Integer, . . .)

Type System

Type System

- all types inherit from an base classes
- Java code automatically generated via JCasGen
- central XML declaration of all types make global reuasibility and exchange between moduls possible
- annotation base class defines character-based begin and end offsets
- types can inherit from other types
- type system as a hierarchical representation of annotations

Type System
Illustration

Example: (from CL)

- a type system might define a token annotation with lemma and part-of-speech feature
- additionally it defines a dependency annotation
- head and dependent of the dependency annotation are token annotations again
- sentence annotations are independent of other annotations

Type System Example

- Token
  - begin
  - end
  - lemma
  - pos
- Sentence
  - begin
  - end
- Dependency
  - begin
  - end
  - dep-token
  - head-token

Type System

Setting up a type system

1. TypeSystem descriptor (like other descriptors) in *src/main/resources*
2. *New* → *Other* ... → *UIMA* → *Type System Descriptor File*
3. add types and features via Component Descriptor Editor[4]

[4]UIMA Plugin

Demonstration

define Token type, JCasGen

CAS

CAS

- Common Analysis Structure
- container for all meta data (primarily annotations) of one observation
- standoff annotations to not modify original document
- associated with a type system
- incrementally enriched by analysis components

CAS

CAS

- components:
  - artifact name
  - document language
  - document text (observation)
  - AnnotationIndex
  - (meta indeces)
- data structure can be nested

CAS Views/Sofas

- possibility of having a nested CAS data structure
- a CAS can have multiple “views”
- view: a perspective on/a facet of an artifact (subject of analysis, sofa)
- views form one CAS, but are perspectives on same data/artifact
- Example: reading comprehension evaluation [Meurers et al., 2011]
  - question view
  - gold standard answer view
  - student answer view

CollectionReader
CAS

- JCas: Java CAS
- wrapper around a CAS with Java functions to work with
- JCas as a Java object with getters/setters and utility functions
- UIMAContext: a static, public object with all the analysis configuration of the application, shared across all CASes

Analysis Engine

- Analysis Engine (AE)
- adds analyses (meta data) of a specific kind to a CAS
- can reuse all previous analyses made by other analysis engines (defined in descriptor which types it relies on)
- modular component, performing exactly one specific analysis step

Analysis Engine

AEs as data-driven components:

- type of action performed depends on the input the component receives
- based on document text and potential previous annotations
- [Gotz and Suhre, 2004, p. 479]: AEs as *"producers of data for downstream components and as consumers of data from upstream components"*

Analysis Enginge Setup

Add two files:

1. Java
   File → *New* → *Class*
   extends
   `org.apache.uima.analysis_component.JCasAnnotator_ImplBase`
2. XML
   New → *Other ...* → *UIMA* → *Analysis Engine Descriptor File*

Analysis Engine Setup

Configuration

- link Java file in descriptor
- import type system
- set input and output capabilities
- add configuration parameters

Configuration Parameters

- possibility to pass parameters to a UIMA component, e.g. language-specific model file or output directory
- access external resources via `InputStream`
- define via ComponentDescriptorEditor
- access values:

```
(String)getConfigParameterValue("outputDir")
```

```
aContext.getResourceAsStream((String)
aContext.getConfigParameterValue(RESOURCE_KEY))
```

initialize/process

- UIMA Java components ususally implement/override `initialize`[5] and `process` method
- `initialize`: only executed once, useful for loading models etc.
- `process`: executed for every CAS in the process

[5]optional

DocumentAnalyzer

- DocumentAnalyzer: pre-fabricated UIMA CPE
- CollectionReader and CASConsumer predefined (files in directory read, write output to XMI and open AnnotationViewer)
- add Java run configuration in Eclipse:
  `org.apache.uima.tools.docanalyzer.DocumentAnalyzer`

Demonstration

AE and DocAnalyzer: primitive whitespace tokenizer

Exercise

Exercise 1-3 (see handout):

1. set up a Java Maven project with UIMA dependencies
2. define types and generate Java code form it
3. wrap the OpenNLP sentence detector in an analysis engine and test it with the DocumentAnalyzer

David Ferrucci and Adam Lally. Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348, 2004.

Thilo Gotz and Oliver Suhre. Design and implementation of the uima common analysis system. *IBM Systems Journal*, 43(3): 476–489, 2004.

Detmar Meurers, Ramon Ziai, Niels Ott, and Janina Kopp. Evaluating answers to reading comprehension questions in context: Results for german and the role of information structure. In *Proceedings of the TextInfer 2011 Workshop on Textual Entailment*, pages 1–9. Association for Computational Linguistics, 2011.