How is this tutorial organized?

Two hour session consisting of

1. A brief introduction about NLP frameworks and libraries
2. A brief introduction to UIMA (non-programming) end-users
3. A brief comparison between UIMA & GATE
4. Some final practical (and programming) UIMA tips

Outline

NLP software

We can classify these tools/libraries/frameworks

- Tools: main program
- "Common Data Representation" (e.g. XML)
- Libraries (APIs): We can use severall funcions (we need to program)
- Frameworks: high level common API + method to integrate new tools + some extra tools (e.g. to avoid programming)

NLP Tools, libraries and frameworks

Which one should I use?
There are many tools, libraries, frameworks for NLP:

- UIMA
- GATE
- Nooj
- Freeling
- OpenNLP
- ...

Which NLP tools/lib/framework should I use

No magical solution, but some important questions are:

- Programming Knowledge - Languages
- Available modules (p.e. we need a parser)
- Group or Individual?
- Flexible or Fix Pipeline?
  - Flexible: use a framework
  - Fix: consider using libraries
- Speed requierements? (frameworks are fast but you pay some overhead)
- Scalability? need grid computing?

Scaling up - Cloud computing

- UIMA Asynchronous Scaleout (Vinci Services instead of Collection Processing Manager)
- GATE on the cloud starting project http://gatecloud.net/
- GATE/UIMA on hadoop, Behemoth 0.1 (not working on hadop 0.20)
- NLTK and Hadoop Streaming (http://www.cloudera.com/blog/2010/03/natural-language-processing-with-hadoop-and-python/)
- ...

Using a common data representation

- All the modules share (an xml) data representation
- Advantages: Loose coupling
- Drawbacks: read / write verbose data representation
- Examples:
  - Living Knowledge Testbed (European project http://livingknowledge-project.eu/)
  - Kyoto project Kyoto core (KAF)

API Based frames

- Tight coupling
- APIs (sometimes accessible from different programming languages)
- Relatively easy to use (for computer scientists)
- Faster
- BUT more complicated to integrate new modules, change the execution flow...

API Based frames

The are many API based libraries / frameworks:

- Open NLP
- Freeling
- TANL
- ...
- NLTK
- Nooj

Open NLP

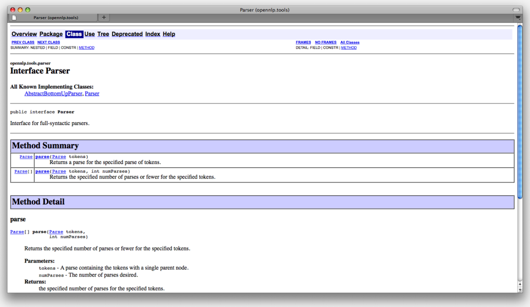
http://opennlp.sourceforge.net/
The opennlp project is a set of java-based NLP tools:

- sentence detection
- tokenization
- pos-tagging
- chunking
- parsing
- named-entity detection
- coreference.

it has been wrapped into GATE, UIMA

Open NLP

API (http://opennlp.sourceforge.net/api/index.html


Parser (opennlp.tools)
Parser (opennlp.tools)
Overview Package Class Use Tree Deprecated Index Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

opennlp.tools.parser
Interface Parser

All Known Implementing Classes:
AbstractBottomUpParser, Parser

```
public interface Parser
```

Interface for full-syntactic parsers.

Method Summary

Parse	parse(Parse tokens) Returns a parse for the specified parse of tokens.
Parse[]	parse(Parse tokens, int numParses) Returns the specified number of parses or fewer for the specified tokens.

Method Detail

parse

```
Parse[] parse(Parse tokens,
              int numParses)
```

Returns the specified number of parses or fewer for the specified tokens.

Parameters:
tokens - A parse containing the tokens with a single parent node.
numParses - The number of parses desired.
Returns:
the specified number of parses for the specified tokens.

Freeling

Freeling is an open source set of NLP tools and resources:

- Spanish, Catalan, English, Italian, Galician, Welsh, Portuguese, Asturian.
- WN-based semantic information access
- word sense disambiguation.
- Expressive rule language for dependency parsing
- Coreference resolution
- C++ / Java / Python API

http://www.lsi.upc.edu/˜nlp/freeling/

TANL

Tanl (Text Analytics and Natural Language) is a suite of modules for text analytics and Natural Language processing.

- Sentence Splitter
- Tokenizer
- POS Tagger, Lemmatizer, Morph tagging
- NE tagger
- Anaphora Resolution
- SuperSense Tagger
- Parser
- Additional tools are used to annotate or process the data: Wikipedia Extractor, Indexer, Dependency Graph Annotator, Hadoop
- C++ / Python API

http://medialab.di.unipi.it/wiki/Tanl/

TANL

Tanl modules can be connected in a pipeline where each module consumes a stream of input items and produces a stream for later modules.
The pipeline model consists of three types of components:

- source: creates an initial pipe (e.g. a document reader)
- transform: receives data from one pipe and produces output on another pipe
- sink: consumes the output of a pipe.

TANL

For example a SentenceSplitter is a source that creates a pipe from an input stream:

```
ss = SentenceSplitter('italian.pickle').pipe(stdin)
```

The pipe can be connected to other stages that perform tokenization, POS tagging and parsing as follows:
$wt = Tokenizer().pipe(ss)$
$pt = PosTagger('italian.pos').pipe(wt)$
$pa = Parser.create('italian.SVM').pipe(pt)$

TANL

A sink is a process that consumes the output of a pipeline, driving the pipeline. It can be as simple as a Python iteration:

```
for sent in pa:
  print sent
```

NLTK

Natural Language Toolkit (NLTK) http://www.nltk.org/

- NLTK was originally created in 2001 as part of a computational linguistics course (University of Pennsylvania)
- Active developing
- Adopted in courses in dozens of universities, and serves as the basis of many research projects

NLTK

- **Simplicity**: To provide an intuitive framework with building blocks
- **Consistency**: To provide a uniform framework with consistent interfaces and data structures
- **Extensibility**: To provide a structure into which new software modules can be easily accommodated (alternative implementations)
- **Modularity**: To provide components that can be used independently without needing to understand the rest of the toolkit

Nooj

Nooj http://www.nooj4nlp.net/

- NooJ is a linguistic development environment that includes large-coverage dictionaries and grammars, and parses corpora in real time.
- NooJ includes tools to create and maintain large-coverage lexical resources,morphological and syntactic grammars.
- NooJ can build complex concordances, with respect to all types of Finite State and Context-Free patterns.
- NooJ helps develop extractors to identify semantic units, such as names of persons, locations, dates, etc.
- Implemented in .NET
- export annotations in xml

Reference Material

These slides are a guide to learn the basics of UIMA. They are not intended to be used as reference material. For reference material, please refer to the official UIMA documentation at:

`http://uima.apache.org/documentation.html`

For an extended tutorial and example applications see ESSLLI 2010 workshop *Painless NLP Programming with UIMA* (with Bard Mellebeck).

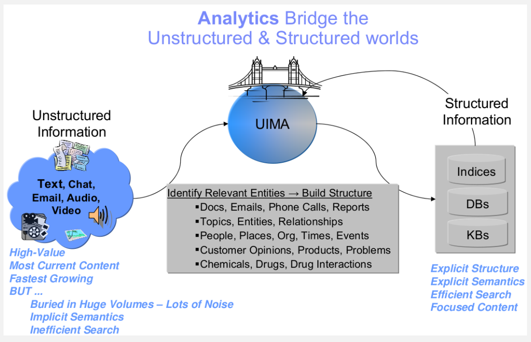
- Course material and exercise code is available at http://esslli2010cph.info/?p=279
- Windows patch available at http://jordi.atserias.cat/home/esslli2010

Outline

What is UIMA?

- UIMA = **U**nstructured **I**nformation **M**anagement **A**rchitecture.
- Unstructured information (text, audio, video, images) >> structured information (e.g. DB).
- UIMA is a software architecture that supports creating, composing and deploying a broad range of analysis capabilities to turn unstructured information into structured information.
- Provides a common data structure (CAS) that is shared between UIMA modules.
- Facilitates sharing of applications between different teams.

What is UIMA?


Analytics Bridge the
Unstructured & Structured worlds
UIMA
Unstructured Information
Text, Chat, Email, Audio, Video
High-Value
Most Current Content
Fastest Growing
BUT ...
Buried in Huge Volumes – Lots of Noise
Implicit Semantics
Inefficient Search
Identify Relevant Entities → Build Structure
▪Docs, Emails, Phone Calls, Reports
▪Topics, Entities, Relationships
▪People, Places, Org, Times, Events
▪Customer Opinions, Products, Problems
▪Chemicals, Drugs, Drug Interactions
Structured Information
Indices
DBs
KBs
Explicit Structure
Explicit Semantics
Efficient Search
Focused Content

What is UIMA?

- The UIMA framework provides a run-time environment in which developers can plug in their UIMA component implementations and with which they can build and deploy UIM applications.
- The UIMA Software Development Kit (SDK) includes the UIMA framework, plus tools and utilities for using UIMA.

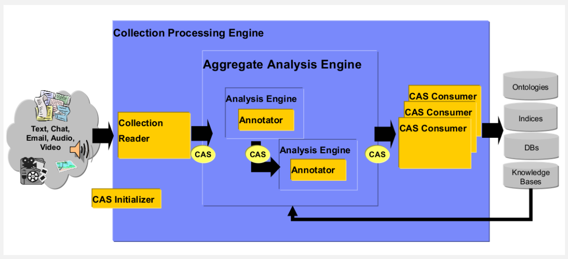
What is UIMA?

- Originally IBM, now Apache project.
- Industrial-strength applications.
- Academic interest:
  - University of Jena, Germany (`www.julielab.de`)
  - Carnegie Mellon University (`uima.lti.cs.cmu.edu`)
  - University of Colorado (`bionlp-uima.sourceforge.net`)
  - University of Tokyo (`www-tsujii.is.s.u-tokyo.ac.jp`)
  - ...
- Apache License.

UIMA = Code + Metadata

- Each UIMA component requires two parts for its implementation:
  1. The declarative part = metadata (in XML).
  2. The actual code (for now: Java and C++, plus support for scripting languages Perl, Python, TCL).
- Metadata describes the component, its identity, structure and behavior.
- This declarative part is called **component descriptor**.

UIMA Building Blocks



Practical Part: What do you need before we can start?

- Java version 1.6.
- UIMA version 2.3.0.
- Eclipse IDE (preferably version 3.6 Helios).
- uimaj-examples (eclipse project)
- Cf. install.pdf for more details.

Type System

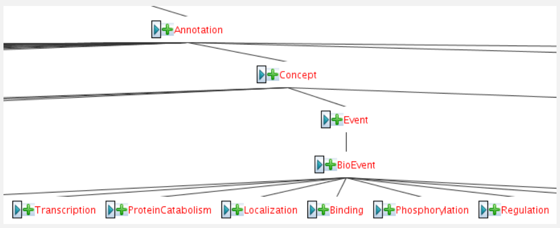
- The Type System determines what annotations the CAS can contain.
- A Type System **is domain and application specific**.

Type System

The built-in Annotation type declares three Features:

- **begin** and **end** store the character offsets of the span of text to which the annotation refers.
- **sofa** (Subject of Analysis) indicates which document the begin and end offsets point to.

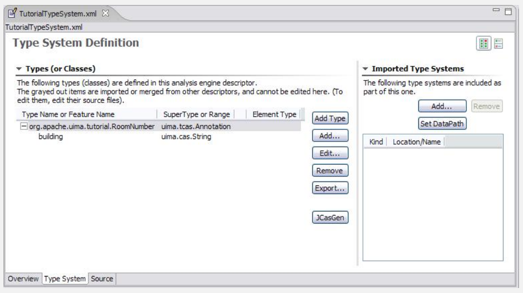
Types are organized hierarchically



Type System

- In Eclipse, expand the uimaj-examples project in the Package Explorer view.
- Right-click on the file in the navigator and select Open With → Component Descriptor Editor.
- Once the editor opens, click on the 'Type System' tab at the bottom of the editor window.
- Try this with the file `uimaj-examples/descriptors/tutorial/ex1/TutorialTypeSystem.xml`.

Type System


TutorialTypeSystem.xml

TutorialTypeSystem.xml

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 Type
org.apache.uima.tutorial.RoomNumber	uima.tcas.Annotation	
building	uima.cas.String	

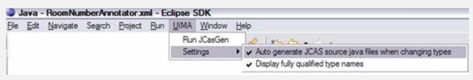
Imported Type Systems

The following type systems are included as part of this one.

Kind	Location/Name

JCasgen

- When you save a descriptor that you have modified, the Component Descriptor Editor will automatically generate Java classes corresponding to the types that are defined in that descriptor (unless this has been disabled), using a utility called JCasGen.
- These Java classes will have the same name (including package) as the CAS types, and will have get and set methods for each of the features that you have defined.


Java - RoomNumberAnnotator.xml - Eclipse SDK

✓ Auto generate JCAS source java files when changing types
✓ Display fully qualified type names

"Block building" UIMA User Case

- We have a set of UIMA components (e.g. tokenizer, parser).
- Based on the same or compatible Type systems.
- Need to select and connect the modules in the appropriate order.
- Set some parameters of the components.

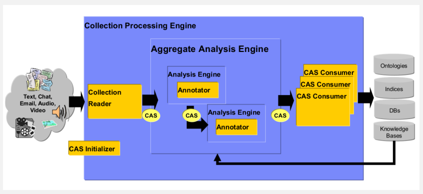
UIMA allows you to do this without programming, by just writing meta data (an Analysis Engine Descriptor).

Analysis Engine

- **Analysis Engine (AE)**: fundamental processing component for document analysis in UIMA.
- Analysis Engines consist of metadata (Descriptor) + code (Annotator).
- E.g. a POS tagger AE consists of an XML descriptor (metadata, parameters, ...) + an annotator implemented in Java.
- Analysis Engines can be **primitive** or **aggregate**.
  - Primitive AEs consist of a single processing module.
  - Aggregate AEs consist of multiple processing modules.

Analysis Engine

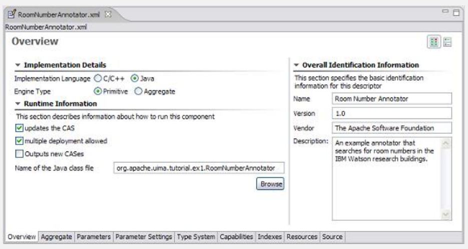
- Gets input information from a CAS (e.g. the text of the document)
- Modify the CAS adding new annotations (or modifying the existing ones)



Analysis Engine Descriptor

- In Eclipse, right-click on `descriptors/tutorial/ex1/RoomNumberAnnotator.xml`
- Open With → Component Descriptor Editor.
- Tip: In Eclipse, you can double click on the tab at the top of the Component Descriptor Editor's window identifying the currently selected editor, and the window will Maximize. Double click it again to restore the original size.

Analysis Engine Descriptor


RoomNumberAnnotator.xml

RoomNumberAnnotator.xml

Overview

Implementation Details

Implementation Language ○ C/C++ [✓ Java]

Engine Type [✓ Primitive] ○ Aggregate

Runtime Information

This section describes information about how to run this component

☑ updates the CAS
☑ multiple deployment allowed
☐ Outputs new CASes

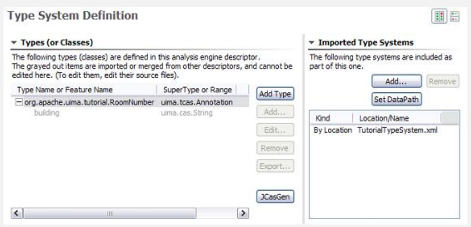
Name of the Java class file: org.apache.uima.tutorial.ex1.RoomNumberAnnotator

Overall Identification Information

This section specifies the basic identification information for this descriptor

Name: Room Number Annotator
Version: 1.0
Vendor: The Apache Software Foundation
Description: An example annotator that searches for room numbers in the IBM Watson research buildings.

Analysis Engines need a Type System


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
org.apache.uima.tutorial.RoomNumber	uima.tcas.Annotation
building	uima.cas.String

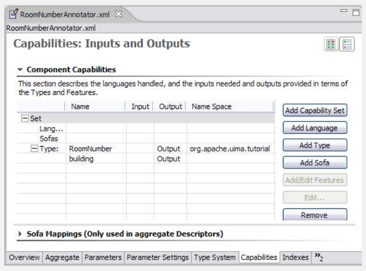
Imported Type Systems

The following type systems are included as part of this one.

Kind	Location/Name
By Location	TutorialTypeSystem.xml

Annotator Capabilities

On the Capabilities page, we define the annotator's inputs and outputs, in terms of the types in the type system.


RoomNumberAnnotator.xml

RoomNumberAnnotator.xml

Capabilities: Inputs and Outputs

Component Capabilities

This section describes the languages handled, and the inputs needed and outputs provided in terms of the Types and Features.

	Name	Input	Output	Name Space
Set				
Lang...				
Sofas				
Type:	RoomNumber		Output	org.apache.uima.tutorial
	building		Output	

Apache UIMA SDK

The UIMA SDK contains a number of tools that facilitate the construction/analysis/debugging of UIM applications.

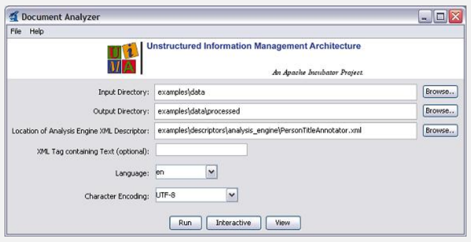
- UIMA Plugins (Edit xml description files)
- Component Descriptor Editor
- Document Analyzer
- Annotation Viewer
- CPE Configurator
- CAS Visual Debugger
- ...

Using the Document Analyzer

In order to run document analyzer from Eclipse:

- Run → Run configurations.
- if UIMA Document Analyzer is not in the Run configurations:
  - Create a new Launch configuration (click icon in left top corner).
  - **Project** Select the UIMA examples project.
  - **Main class** Select the class Document Analyzer (org.apache.uima.tools)

Document Analyzer


Document Analyzer

Unstructured Information Management Architecture

An Apache Incubator Project

Input Directory: examples\data

Output Directory: examples\data\processed

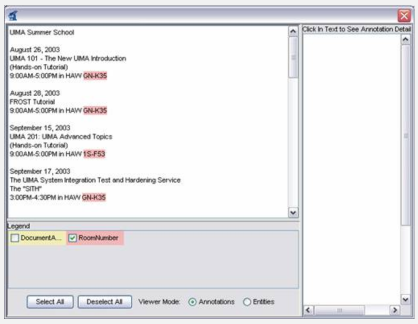
Location of Analysis Engine XML Descriptor: examples\descriptors\analysis_engine\PersonTitleAnnotator.xml

XML Tag containing Text (optional):

Language: en

Character Encoding: UTF-8

Document Analyzer Results


UIMA Summer School

August 26, 2003
UIMA 101 - The New UIMA Introduction
(Hands-on Tutorial)
9:00AM-5:00PM in HAW GN-K35

August 28, 2003
FROST Tutorial
9:00AM-5:00PM in HAW GN-K35

September 15, 2003
UIMA 201: UIMA Advanced Topics
(Hands-on Tutorial)
9:00AM-5:00PM in HAW 1S-F53

September 17, 2003
The UIMA System Integration Test and Hardening Service
The "SITH"
3:00PM-4:30PM in HAW GN-K35

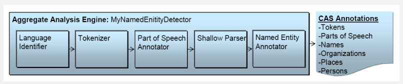
Click In Text to See Annotation Detail

Legend: ☐ DocumentA... ☑ RoomNumber

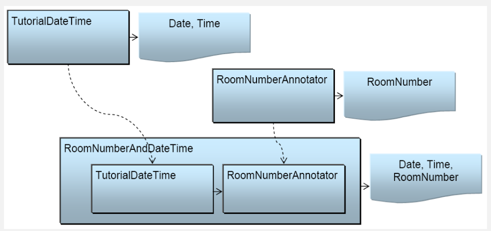
Viewer Mode: [✓ Annotations] Entities

Aggregate Analysis Engine: basic principle

- A Single Analysis Engine reads in the CAS, does some processing and updates the CAS.
- An Aggregate Analysis Engine is a unit that contains multiple Analysis Engines.
- Aggregate Analysis Engines are built to encapsulate a potentially complex internal structure and insulate it from users of the AE.

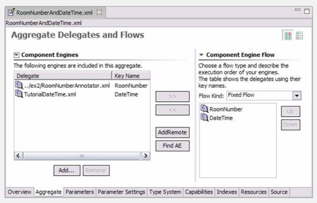


Example: Combining Several Analysis Engines



Aggregate Analysis Engine Descriptor

Open `RoomNumberAndDateTime.xml` in `descriptors/tutorial/ex3` in the uimaj-examples project.


RoomNumberAndDateTime.xml

RoomNumberAndDateTime.xml

Aggregate Delegates and Flows

Component Engines

The following engines are included in this aggregate.

Delegate	Key Name
../ex2/RoomNumberAnnotator.xml	RoomNumber
TutorialDateTime.xml	DateTime

Component Engine Flow

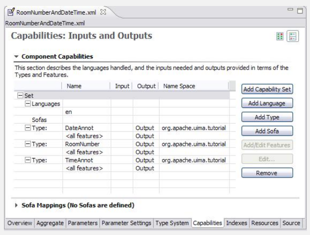
Choose a flow type and describe the execution order of your engines. The table shows the delegates using their key names.

Flow Kind: Fixed Flow

RoomNumber
DateTime

Aggregate Analysis Engine Capabilities

You explicitly declare the aggregate Analysis Engine's inputs and outputs on the Capabilities page:



RoomNumberAndDateTime.xml

Capabilities: Inputs and Outputs

Component Capabilities

This section describes the languages handled, and the inputs needed and outputs provided in terms of the Types and Features.

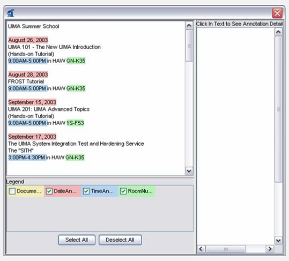
	Name	Input	Output	Name Space
Set				
Languages				
	en			
Sofas				
Type:	DateAnnot		Output	org.apache.uima.tutorial
	<all features>		Output	
Type:	RoomNumber		Output	org.apache.uima.tutorial
	<all features>		Output	
Type:	TimeAnnot		Output	org.apache.uima.tutorial
	<all features>		Output	

Sofa Mappings (No Sofas are defined)

Add Capability Set
Add Language
Add Type
Add Sofa
Add/Edit Features
Edit...
Remove
Overview | Aggregate | Parameters | Parameter Settings | Type System | Capabilities | Indexes | Resources | Source

Aggregate AEs behave the same as Primitive AEs

- Run → Run configurations → Document Analyzer
- Select descriptor `examples/descriptors/tutorial/ex3/RoomNumberAndDateTime.xml`



UIMA Summer School

August 26, 2003
UIMA 101 - The New UIMA Introduction
(Hands-on Tutorial)
9:00AM-5:00PM in HAW GN-K35

August 28, 2003
FROST Tutorial
9:00AM-5:00PM in HAW GN-K35

September 15, 2003
UIMA 201: UIMA Advanced Topics
(Hands-on Tutorial)
9:00AM-5:00PM in HAW 1S-F53

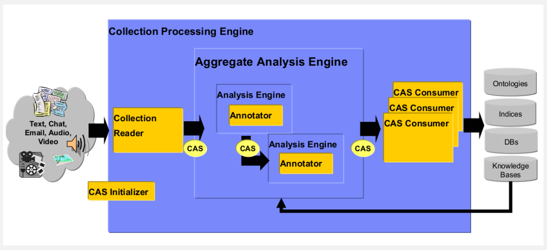
September 17, 2003
The UIMA System Integration Test and Hardening Service
The "SITH"
3:00PM-4:30PM in HAW GN-K35

Some hints about flow and scalability

- Workflows:
  - Fixed Flow: pipeline
  - Capability Language Flow: skip AE base on Language / output capability
- User can implement their own workflows (code + descriptors)
- e.g. org.apache.uima.examples.flow.WhiteboardFlowController

Collection Processing

- Many UIMA applications analyze entire collections of documents.
- CPE = Collection Processing Engine.

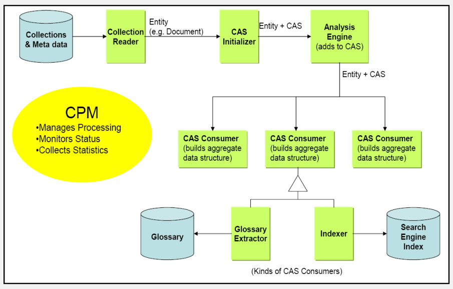


Collection Processing Engine (CPE)

A CPE has 3 main components:

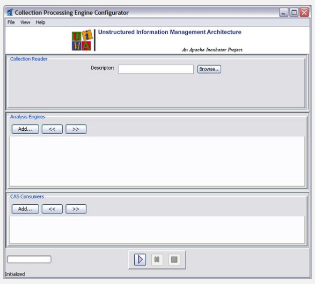
- Collection Readers
- Analysis Engine
- CasConsumer

CPE Components



CPE Configurator

Run → Run configurations → UIMA CPE GUI.


Collection Processing Engine Configurator
File View Help
Unstructured Information Management Architecture
An Apache Incubator Project
Collection Reader
Descriptor:
Browse..
Analysis Engines
Add... << >>
CAS Consumers
Add... << >>
Initialized

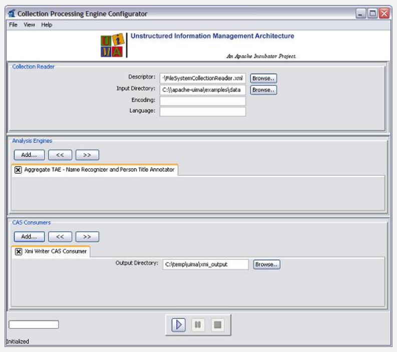
CPE example: set the CPE Configurator

Collection Reader:
UIMA_HOME/examples/descriptors/
collection_reader/FileSystemCollectionReader.xml

Analysis Engine:
UIMA_HOME/examples/descriptors/
analysis_engine/namesAndPersonTitles_TAE.xml

CAS Consumer:
UIMA_HOME/examples/descriptors/
cas_consumer/XmiWriterCasConsumer.xml

CPE example


Collection Processing Engine Configurator

File View Help

Unstructured Information Management Architecture

An Apache Incubator Project.

Collection Reader

Descriptor: ·\FileSystemCollectionReader.xml Browse..

Input Directory: C:\\apache-uima\examples\data Browse..

Encoding:

Language:

Analysis Engines

Add... << >>

☒ Aggregate TAE - Name Recognizer and Person Title Annotator

CAS Consumers

Add... << >>

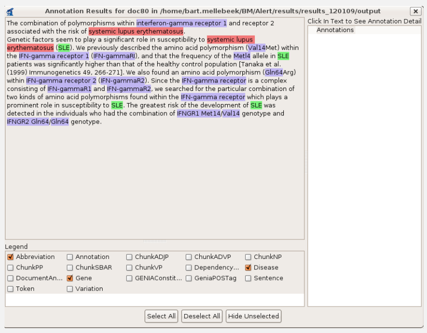
☒ Xmi Writer CAS Consumer

Output Directory: C:\temp\uima\xmi_output Browse..

Initialized

Annotation Viewer

- Viewer for exploring annotations and related CAS data.


Annotation Results for doc80 in /home/bart.mellebeek/BM/Alert/results/results_120109/output

The combination of polymorphisms within interferon-gamma receptor 1 and receptor 2 associated with the risk of systemic lupus erythematosus.
Genetic factors seem to play a significant role in susceptibility to systemic lupus erythematosus (SLE). We previously described the amino acid polymorphism (Val14Met) within the IFN-gamma receptor 1 (IFN-gammaRl), and that the frequency of the Metl4 allele in SLE patients was significantly higher than that of the healthy control population [Tanaka et al. (1999) Immunogenetics 49, 266-271]. We also found an amino acid polymorphism (Gln64Arg) within IFN-gamma receptor 2 (IFN-gammaR2). Since the IFN-gamma receptor is a complex consisting of IFN-gammaR1 and IFN-gammaR2, we searched for the particular combination of two kinds of amino acid polymorphisms found within the IFN-gamma receptor which plays a prominent role in susceptibility to SLE. The greatest risk of the development of SLE was detected in the individuals who had the combination of IFNGR1 Met14/Val14 genotype and IFNGR2 Gln64/Gln64 genotype.

Click In Text to See Annotation Detail

Annotations

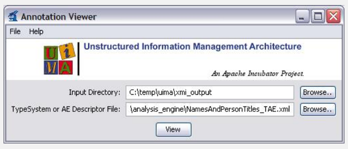
Legend

✓ Abbreviation	☐ Annotation	☐ ChunkADJP	☐ ChunkADVP	☐ ChunkNP
☐ ChunkPP	☐ ChunkSBAR	☐ ChunkVP	☐ Dependency...	✓ Disease
☐ DocumentAn...	✓ Gene	☐ GENIAConstit...	☐ GeniaPOSTag	☐ Sentence
☐ Token	☐ Variation			

Select All | Deselect All | Hide Unselected

Annotation Viewer

- Run → Run configurations → UIMA Annotation Viewer
- Input: examples/data/processed
- TS or descriptor: examples/descriptors/analysis_engine/NamesAndPersonTitles_TAE.xml


Annotation Viewer

File Help

Unstructured Information Management Architecture

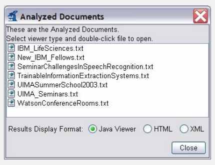
An Apache Incubator Project

Input Directory: C:\temp\uima\xmi_output Browse...

TypeSystem or AE Descriptor File: \analysis_engine\NamesAndPersonTitles_TAE.xml Browse...

View

Annotation Viewer Results

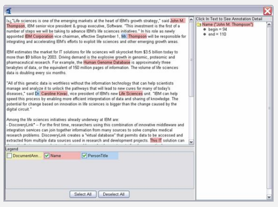

Analyzed Documents

These are the Analyzed Documents.
Select viewer type and double-click file to open.

IBM_LifeSciences.txt
New_IBM_Fellows.txt
SeminarChallengesInSpeechRecognition.txt
TrainableInformationExtractionSystems.txt
UIMASummerSchool2003.txt
UIMA_Seminars.txt
WatsonConferenceRooms.txt

Results Display Format: [✓ Java Viewer] HTML XML

Close


ï»¿"Life sciences is one of the emerging markets at the heart of IBM's growth strategy," said John M. Thompson, IBM senior vice president & group executive, Software. "This investment is the first of a number of steps we will be taking to advance IBM's life sciences initiatives." In his role as newly appointed IBM Corporation vice chairman, effective September 1, Mr. Thompson will be responsible for integrating and accelerating IBM's efforts to exploit life sciences and other emerging growth areas.

IBM estimates the market for IT solutions for life sciences will skyrocket from $3.5 billion today to more than $9 billion by 2003. Driving demand is the explosive growth in genomic, proteomic and pharmaceutical research. For example, the Human Genome Database is approximately three terabytes of data, or the equivalent of 150 million pages of information. The volume of life sciences data is doubling every six months.

"All of this genetic data is worthless without the information technology that can help scientists manage and analyze it to unlock the pathways that will lead to new cures for many of today's diseases," said Dr. Caroline Kovac, vice president of IBM's new Life Sciences unit. "IBM can help speed this process by enabling more efficient interpretation of data and sharing of knowledge. The potential for change based on innovation in life sciences is bigger than the change caused by the digital circuit."

Among the life sciences initiatives already underway at IBM are:
- DiscoveryLink* -- For the first time, researchers using this combination of innovative middleware and integration services can join together information from many sources to solve complex medical research problems. DiscoveryLink creates a "virtual database" that permits data to be accessed and extracted from multiple data sources used in research and development projects. This IT solution can

Click In Text to See Annotation Detail

Name ("John M. Thompson")
- begin = 94
- end = 110

Legend

☐ DocumentAnn... ✓ Name ✓ PersonTitle

Select All | Deselect All

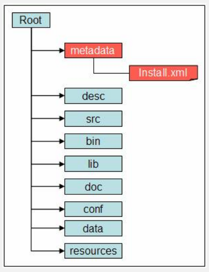
PEAR file

- A PEAR (Processing Engine ARchive) file is a standard package for UIMA components.
- The PEAR package can be used for distribution and reuse by other components or applications.
- It also allows applications and tools to manage UIMA components automatically for verification, deployment, invocation, testing, etc.
- Can be generated inside Eclipse or using maven plugin

The PEAR Structure

Important folders

- `metadata/` contains the PEAR installation descriptor `install.xml`.
- `desc/` contains descriptor files of analysis engines, delegates analysis engines (all levels), and other components (Collection Readers, CAS Consumers, etc).



PEAR files must be self-contained

- The PEAR structure must be self-contained: the component must run properly independently from the PEAR root folder location.
- No absolute paths.
- Also handy if applications in a UIMA pipeline use different versions of a specific jar file.

The `install.xml` file

- <SUBMITTED_COMPONENT>: the component id is a string that uniquely identifies the component.
- <INSTALLATION>: this section specifies the external dependencies of the component and the operations that should be performed during the PEAR package installation.
- The <INSTALLATION> section may specify the following operations:
  - Setting environment variables that are required to run the installed component.
  - Finding and replacing string expressions in files.
- For more info, Cf.
  `http://uima.apache.org/downloads/releaseDocs/2.3.0-incubating/docs/html/references/references.ht`

Macros

- Macros are variables that can be used to represent the full path of a certain directory during or after installation. The use of macros make PEAR files generic enough so they can be installed everywhere.
- There are two macro variables:
  1. $main_root: e.g. `$main_root/data`
  2. $component_id$root: e.g. `$my.comp.Dictionary$root/resources/dict`
- These variables can be inserted in certain parts of `install.xml` and in files in the `desc/` and `conf/` directories. During installation of the PEAR, they are replaced by their values. These values are stored in the metadata/PEAR.properties file that is generated during PEAR installation.

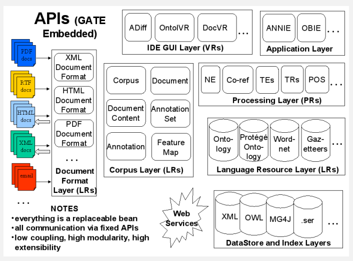
Outline

Gate:General Architecture For Text Engineering

GATE general architecture for text engineering

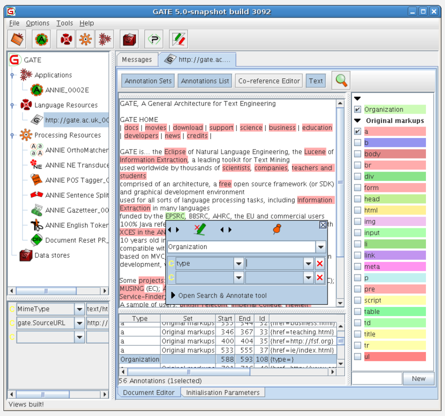
General Architecture For Text Engineering is over 15 years old and is in active use for all types of computational tasks involving human language. http://gate.ac.uk

Gate:General Architecture For Text Engineering



Refer to GATE first page

Gate:General Architecture For Text Engineering



MimeType text/ht
gate.SourceURL http://

GATE, A General Architecture for Text Engineering

GATE HOME
| docs | movies | download | support | science | business | education
| developers | news | credits |

GATE is... the Eclipse of Natural Language Engineering, the Lucene of Information Extraction, a leading toolkit for Text Mining
used worldwide by thousands of scientists, companies, teachers and students
comprised of an architecture, a free open source framework (or SDK) and graphical development environment
used for all sorts of language processing tasks, including Information Extraction in many languages
funded by the EPSRC, BBSRC, AHRC, the EU and commercial users
100% Java refe
XCES in the AN
10 years old in
compatible wit
based on MVC
development,

Some projects
MUSING (EC);
Service-Finder
A sample of us

Organization
type

Type	Set	Start	End	Id	
a	Original markups	346	367	33	{href=teaching.html}
a	Original markups	400	404	35	{href=http://fsf.org}
a	Original markups	533	555	37	{href=ie/index.html}
Organization		588	593	108	{type=}

56 Annotations (1selected)

Gate:General Architecture For Text Engineering

GATE components are one of three types:

- **LanguageResources (LRs)** represent entities such as lexicons, corpora or ontologies;
- **ProcessingResources (PRs)** represent entities that are primarily algorithmic, such as parsers, generators or ngram modellers;
- **VisualResources (VRs)** represent visualisation and editing components that participate in GUIs.

Gate:General Architecture For Text Engineering

Metadata (xml) to configure your resources, e.g. creole.xml file

```xml
<CREOLE-DIRECTORY>
<CREOLE>
  <RESOURCE>
    <NAME>Minipar Wrapper</NAME>
    <JAR>MiniparWrapper.jar</JAR>
    <CLASS>minipar.Minipar</CLASS>
    <COMMENT>MiniPar is a shallow parser. It determines the
    dependency relationships between the words of a sentence.</COMMENT>
    <HELPURL>http://gate.ac.uk/cgi-bin/userguide/sec:parsers:minipar</HELPURL>
    <PARAMETER NAME="document"
  RUNTIME="true"
  COMMENT="document to process">gate.Document</PARAMETER>
    <PARAMETER NAME="miniparDataDir"
        RUNTIME="true"
        COMMENT="location of the Minipar data directory">
        java.net.URL
    </PARAMETER>
    <PARAMETER NAME="miniparBinary"
        RUNTIME="true"
        COMMENT="Name of the Minipar command file">
        java.net.URL
```

Gate and UIMA

- In GATE, unit of processing is the Document
  - Text, plus features, plus annotations
  - Annotations can have arbitrary features, with any Java object as value
- In UIMA, unit of processing is (T)CAS (common analysis structure)
  - Text, plus Feature Structures
  - Annotations are just a special kind of FS, which includes start and end offset features

Gate and UIMA

- In GATE, annotations can have any features, with any values
- In UIMA, feature structures are strongly typed
  - Must declare what types of annotations are supported by each analysis engine
  - Must specify what features each annotation type supports
  - Must specify what type feature values may take
  - Primitive types - string, integer, float
  - Reference types - reference to another FS in the CAS
  - Arrays of the above
  - All defined in XML descriptor for the AE

Complex typesystem

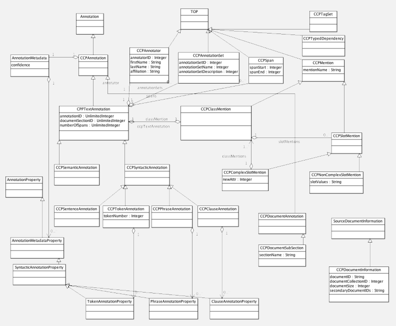


Figure: BioNLP-UIMA UML Diagram

Complex typesystem

- Wrappers depend on types
- Different types for the same kind of information
- opennlp.uima.Token vs de.julielab.jules.types.Token
- Parametrization of types in the wrapper can easy this problem (see opennlp-uima)

UIMA mixing C++ and Java

UIMA c++ is still quite new
http://uima.apache.org/doc-uimacpp-huh.html

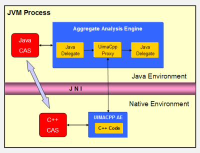


Figure: UIMA C++ and Java AE

Gate and UIMA

More Information about GATE and UIMA

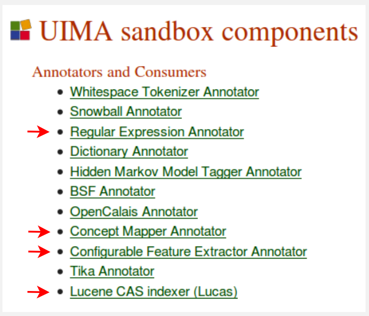
- GATE DOC http://gate.ac.uk/sale/tao/splitch18.html#chap:uima
- talk http://gate.ac.uk/g8/page/print/2/sale/talks/gate-course.../uimaintegration.ppt
- talk http://gate.ac.uk/sale/talks/gatecoursejuly09/slidespdf/uima-integration.pdf

Outline

What is the Sandbox?

- A workspace that is open to all UIMA committers and developers who would like to contribute code and join the UIMA developer community.
- `http://uima.apache.org/sandbox.html`

UIMA Sandbox Components



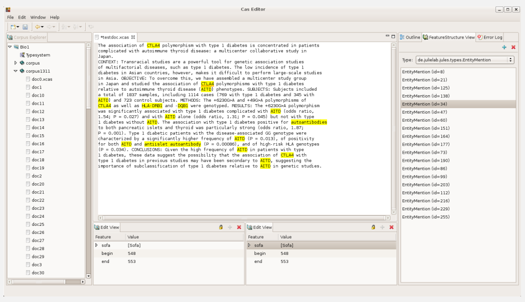
UIMA sandbox components

Annotators and Consumers

- Whitespace Tokenizer Annotator
- Snowball Annotator
- Regular Expression Annotator
- Dictionary Annotator
- Hidden Markov Model Tagger Annotator
- BSF Annotator
- OpenCalais Annotator
- Concept Mapper Annotator
- Configurable Feature Extractor Annotator
- Tika Annotator
- Lucene CAS indexer (Lucas)

CAS Editor: Manual Annotations with UIMA

`http://incubator.apache.org/uima/sandbox.html`



*testdoc.xcas

```
The association of CTLA4 polymorphism with type 1 diabetes is concentrated in patients
complicated with autoimmune thyroid disease: a multicenter collaborative study in
Japan.
CONTEXT: Transracial studies are a powerful tool for genetic association studies
of multifactorial diseases, such as type 1 diabetes. The low incidence of type 1
diabetes in Asian countries, however, makes it difficult to perform large-scale studies
in Asia. OBJECTIVE: To overcome this, we have assembled a multicenter study group
in Japan and studied the association of CTLA4 polymorphisms with type 1 diabetes
relative to autoimmune thyroid disease (AITD) phenotypes. SUBJECTS: Subjects included
a total of 1837 samples, including 1114 cases (769 with type 1 diabetes and 345 with
AITD) and 723 control subjects. METHODS: The +6230G>A and +49G>A polymorphisms of
CTLA4 as well as HLA-DRB1 and -DQB1 were genotyped. RESULTS: The +6230G>A polymorphism
was significantly associated with type 1 diabetes complicated with AITD (odds ratio,
1.54; P = 0.027) and with AITD alone (odds ratio, 1.31; P = 0.045) but not with type
1 diabetes without AITD. The association with type 1 diabetes positive for autoantibodies
to both pancreatic islets and thyroid was particularly strong (odds ratio, 1.87;
P = 0.001). Type 1 diabetic patients with the disease-associated GG genotype were
characterized by a significantly higher frequency of AITD (P = 0.013), of positivity
for both AITD and antiislet autoantibody (P = 0.00086), and of high-risk HLA genotypes
(P = 0.034). CONCLUSIONS: Given the high frequency of AITD in patients with type
1 diabetes, these data suggest the possibility that the association of CTLA4 with
type 1 diabetes in previous studies may have been secondary to AITD, suggesting the
importance of subclassification of type 1 diabetes relative to AITD in genetic studies.
```

Edit View

Feature	Value
sofa	[Sofa]
begin	548
end	553

Edit View

Feature	Value
sofa	[Sofa]
begin	548
end	553

FeatureStructure View

Type: de.julielab.jules.types.EntityMention

EntityMention (id=8)
EntityMention (id=21)
EntityMention (id=125)
EntityMention (id=138)
EntityMention (id=34)
EntityMention (id=47)
EntityMention (id=60)
EntityMention (id=151)
EntityMention (id=164)
EntityMention (id=177)
EntityMention (id=73)
EntityMention (id=190)
EntityMention (id=86)
EntityMention (id=99)
EntityMention (id=203)
EntityMention (id=112)
EntityMention (id=216)
EntityMention (id=229)
EntityMention (id=255)

UIMA Repositories

`uima.lti.cs.cmu.edu`	a repository of UIMA components.
`u-compare.org`	an integrated text mining/natural language processing system, Java Web Start Technology.
`www.julielab.de`	a comprehensive NLP tool suite for the application purposes of semantic search, information extraction and text mining, focused on biomedical text processing.
`bionlp-uima.sourceforge.net`	UIMA wrappers for novel and well-known 3rd-party NLP tools used in biomedical text prosessing.

Write an existing java PoS Tagging into UIMA

- We have a java PoS tagger
- The tagger has some parameters/options (model, tagset, encoding, etc.)
- The tagger given the list of tokens of a sentence returns a list of PoS
- We want to use and existing type system (e.g. opennlp)

Wrapper = Analysis Engine

An Analysis Engine has two parts:

- Description (metadata)
- Annotator (code)

UIMA Description for my PoS Tagging

- Define the input and output (so we need the type system)
- Parameters (default values, ...)
- Java class that will do the job

(Remember ... We have done that in the programming less introduction)

UIMA Description for my PoS Tagging

- Define the input and output (so we need the type system)
- Parameters (default values, ...)
- Java class that will do the job

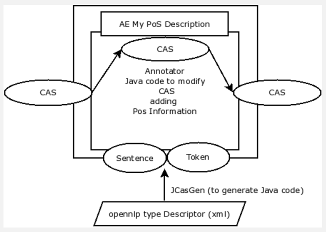
UIMA Annotator for my PoS Tagging

Steps:

- Initialize the PoS tagger (Collect some parameters to create object,loading model)
- Do the tagging
  - Access sentence and token information from UIMA CAS
  - Call the tagger with the tokens
  - Update/Add the Pos Information to the UIMA CAS
- Free resources

UIMA Annotator for my PoS Tagging

- An annotator extends class JCasAnnotator_ImplBase
- Initialization must be done by method **initialize(UimaContext)**
- The annotation process is carried out by the method **process(JCas)**
- Finalize/Free resources must be done by method **destroy()**



UIMA Annotator for my PoS Tagging

```
 // initilize PosTagger processor
@Override
public void initialize(UimaContext context) {
super.initialize(context);
posModel= (String) context.getConfigParameterValue("model");
posTagset=  (String)  context.getConfigParameterValue("tagset");
enc=  (String)  context.getConfigParameterValue("encoding");
    ...
```

Annotations are stored in Indexes

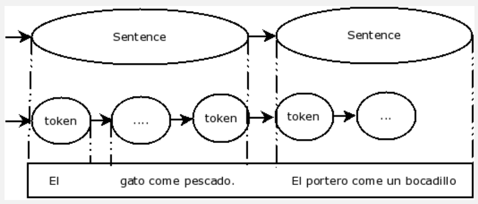
- The results of the annotators are stored in the CAS as annotations.
- Annotations can be accessed using an Index.
- Obtaining an Iterator that allows you to step through all annotations of a particular type.

```
FSIndex sentenceIndex = aJCas.getAnnotationIndex(Sentence.type);
```

Iterators

```
FSIterator SentenceIter = SentenceIndex.iterator();
while (SentenceIter.hasNext()) {
  Sentence Sentence = (Sentence)SentenceIter.next();
  //do something
}
```

UIMA Annotator for my PoS Tagging



- **subiterator**: obtain the Annotations "inside".
- **getCoveredText**: obtain the original text covered by the annotation.

UIMA Annotator for my PoS Tagging

```
public void process(JCas jcas) {
//For every sentence
FSIterator<Annotation> iter_sen =
      jcas.getAnnotationIndex(Sentence.type).iterator();
while(iter_sen.hasNext()) {
   Sentence sen = iter_sen.next();
   //For every Token in a sentence
   FSIterator<Annotation> iter_token =
           jcas.getAnnotationIndex(Token.type).subiterator(sen);
       while (iter_token.hasNext()) {
         Annotation  pt = iter_token.next();
         String wordForm = pt.getCoveredText();
         words.add(wordForm);
       }
                  ...
```

UIMA Annotator for my PoS Tagging

```
  ...
//tag sentence
posTag = posTagger.tagSequence(words);
   ....
```

UIMA Annotator for my PoS Tagging

```
// add PoS to token information
int i=0;
iter_token = jcas.getAnnotationIndex(tokenType).subiterator(sen);
while(iter_token.hasNext()) {
            Token pt = (Token) iter_token.next();
    //set the PoS feature
    pt.setPos(posTag[i++]);
}
           ...
```

Final remarks

Notice that this is a oversimplified example. P.e. Model should be better defined as a resource, annotation types can be parametrized, ... etc.

Glossary

The following slides contain a glossary[1] of the most important concepts covered in this course.

Aggregate Analysis Engine An Analysis Engine made up of multiple subcomponent Analysis Engines arranged in a flow. The flow can be one of the two built-in flows, or a custom flow provided by the user.

Analysis Engine A program that analyzes artifacts (e.g. documents) and infers information about them, and which implements the UIMA Analysis Engine interface Specification. It does not matter how the program is built, with what framework or whether or not it contains component (“sub”) Analysis Engines.

Annotation The association of a metadata, such as a label, with a region of text (or other type of artifact). For example, the label ‘Person’ associated with a region of text ‘John Doe’ constitutes an annotation. We say Person annotates the span of text from X to Y containing exactly ‘John Doe’. An annotation is represented as a special type in a UIMA type system. It is the type used to record the labeling of regions of a Sofa.

Annotator A software component that implements the UIMA annotator interface. Annotators are implemented to produce and record annotations over regions of an artifact (e.g., text document, audio, and video).

[1] source: documentation Apache UIMA.

Glossary

Apache UIMA Java Framework A Java-based implementation of the UIMA architecture. It provides a run-time environment in which developers can plug in and run their UIMA component implementations and with which they can build and deploy UIM applications. The framework is the core part of the Apache UIMA SDK.

Apache UIMA Software Development Kit (SDK) The SDK for which you are now reading the documentation. The SDK includes the framework plus additional components such as tooling and examples. Some of the tooling is Eclipse-based (http://www.eclipse.org/). The Apache UIMA SDK is being developed at the Apache Incubator.

CAS The UIMA Common Analysis Structure is the primary data structure which UIMA analysis components use to represent and share analysis results. It contains:

- The artifact. This is the object being analyzed such as a text document or audio or video stream. The CAS projects one or more views of the artifact. Each view is referred to as a Sofa.
- A type system description – indicating the types, subtypes, and their features.
- Analysis metadata – standoff annotations describing the artifact or a region of the artifact
- An index repository to support efficient access to and iteration over the results of analysis.

Glossary

CAS Consumer A component that receives each CAS in the collection, usually after it has been processed by an Analysis Engine. It is responsible for taking the results from the CAS and using them for some purpose, perhaps storing selected results into a database, for instance. The CAS Consumer may also perform collection-level analysis, saving these results in an application-specific, aggregate data structure.

CAS Processor A component of a Collection Processing Engine (CPE) that takes a CAS as input and returns a CAS as output. There are two types of CAS Processors: Analysis Engines and CAS Consumers.

CAS View A CAS Object which shares the base CAS and type system definition and index specifications, but has a unique index repository and a particular Sofa. Views are named, and applications and annotators can dynamically create additional views whenever they are needed. Annotations are made with respect to one view. Feature structures can have references to feature structures indexed in other views, as needed.

CDE The Component Descriptor Editor. This is the Eclipse tool that lets you conveniently edit the UIMA descriptors.

Glossary

Collection Processing Engine (CPE) Performs Collection Processing through the combination of a Collection Reader, 0 or more Analysis Engines, and zero or more CAS Consumers. The Collection Processing Manager (CPM) manages the execution of the engine.

Collection Processing Manager (CPM) The part of the framework that manages the execution of collection processing, routing CASs from the Collection Reader to 0 or more Analysis Engines and then to the 0 or more CAS Consumers. The CPM provides feedback such as performance statistics and error reporting and supports other features such as parallelization and error handling.

Collection Reader A component that reads documents from some source, for example a file system or database. The collection reader initializes a CAS with this document. Each document is returned as a CAS that may then be processed by an Analysis Engines. If the task of populating a CAS from the document is complex, you may use an arbitrarily complex chain of Analysis Engines and have the last one create and initialize a new Sofa.

Feature A data member or attribute of a type. Each feature itself has an associated range type, the type of the value that it can hold. In the database analogy where types are tables, features are columns. In the world of structured data types, each feature is a ‘field’, or data member.

Glossary

Flow Controller A component which implements the interfaces needed to specify a custom flow within an Aggregate Analysis Engine.

Index Data in the CAS can only be retrieved using Indexes. Indexes are analogous to the indexes that are specified on tables of a database. Indexes belong to Index Repositories; there is one Repository for each view of the CAS. Indexes are specified to retrieve instances of some CAS Type (including its subtypes), and can be optionally sorted in a user-definable way. For example, all types derived from the UIMA built-in type uima.tcas.Annotation contain begin and end features, which mark the begin and end offsets in the text where this annotation occurs. There is a built-in index of Annotations that specifies that annotations are retrieved sequentially by sorting first on the value of the begin feature (ascending) and then by the value of the end feature (descending). In this case, iterating over the annotations, one first obtains annotations that come sequentially first in the text, while favoring longer annotations, in the case where two annotations start at the same offset. Users can define their own indexes as well.

Glossary

JCas A Java object interface to the contents of the CAS. This interface use additional generated Java classes, where each type in the CAS is represented as a Java class with the same name, each feature is represented with a getter and setter method, and each instance of a type is represented as a Java object of the corresponding Java class.

PEAR An archive file that packages up a UIMA component with its code, descriptor files and other resources required to install and run it in another environment. You can generate PEAR files using utilities that come with the UIMA SDK.

Primitive Analysis Engine An Analysis Engine that is composed of a single Annotator; one that has no component (or ‘sub’) Analysis Engines inside of it; contrast with Aggregate Analysis Engine.

Semantic Search search where the semantic intent of the query is specified using one or more entity or relation specifiers. For example, one could specify that they are looking for a person (named) ‘Bush’. Such a query would then not return results about the kind of bushes that grow in your garden but rather just persons named Bush.

Glossary

Structured Information Items stored in structured resources such as search engine indices, databases or knowledge bases. The canonical example of structured information is the database table. Each element of information in the database is associated with a precisely defined schema where each table column heading indicates its precise semantics, defining exactly how the information should be interpreted by a computer program or end-user.

Subject of Analysis (Sofa) A piece of data (e.g., text document, image, audio segment, or video segment), which is intended for analysis by UIMA analysis components. It belongs to a CAS View which has the same name; there is a one-to-one correspondence between these. There can be multiple Sofas contained within one CAS, each one representing a different view of the original artifact – for example, an audio file could be the original artifact, and also be one Sofa, and another could be the output of a voice-recognition component, where the Sofa would be the corresponding text document. Sofas may be analyzed independently or simultaneously; they all co-exist within the CAS.

Glossary

Type A specification of an object in the CAS used to store the results of analysis. Types are defined using inheritance, so some types may be defined purely for the sake of defining other types, and are in this sense ‘abstract types’ Types usually contain Features, which are attributes, or properties of the type. A type is roughly equivalent to a class in an object oriented programming language, or a table in a database. Instances of types in the CAS may be indexed for retrieval.

Type System A collection of related types. All components that can access the CAS, including Applications, Analysis Engines, Collection Readers, Flow Controllers, or CAS Consumers declare the type system that they use. Type systems are shared across Analysis Engines, allowing the outputs of one Analysis Engine to be read as input by another Analysis Engine. A type system is roughly analogous to a set of related classes in object oriented programming, or a set of related tables in a database. The type system / type / feature terminology comes from computational linguistics.

Unstructured Information The canonical example of unstructured information is the natural language text document. The intended meaning of a document’s content is only implicit and its precise interpretation by a computer program requires some degree of analysis to explicate the document’s semantics. Other examples include audio, video and images. Contrast with Structured Information.

Glossary

UIMA UIMA is an acronym that stands for Unstructured Information Management Architecture; it is a software architecture which specifies component interfaces, design patterns and development roles for creating, describing, discovering, composing and deploying multi-modal analysis capabilities.

XCAS An XML representation of the CAS. The XCAS can be used for saving and restoring CASs to and from streams. The UIMA SDK provides XCAS serialization and de-serialization methods for CASes. This is an older serialization format and new UIMA code should use the standard XMI format instead.

XML Metadata Interchange (XMI) An OMG standard for representing object graphs in XML, which UIMA uses to serialize analysis results from the CAS to an XML representation. The UIMA SDK provides XMI serialization and de-serialization methods for CASes.

Links

- Freeling http://www.lsi.upc.edu/~nlp/freeling/
- JULIE Labs http://www.julielab.de
- GATE http://gate.ac.uk
- Living Knowledge Testbed http://livingknowledge-project.eu/)
- NLTK http://www.nltk.org/
- Nooj http://www.nooj4nlp.net/
- OpenNLP http://opennlp.sourceforge.net/
- TANL http://medialab.di.unipi.it/wiki/Tanl/
- UIMA http://uima.apache.org
- UIMA BIONLP http://bionlp-uima.sourceforge.net
- UIMA Component Repository at CMU http://uima.lti.cs.cmu.edu
- UIMA Sandbox http://incubator.apache.org/uima/sandbox.html
- U-Compare http://u-compare.org