UIMA: Unstructured Information Management Architecture

Alessandro Moschitti
Department of Computer Science and Information Engineering
University of Trento
Email: moschitti@disi.unitn.it
Motivations

- Nowadays, natural language processing systems are becoming more and more complex
- Many linguistic processors:
Motivations

- Many formalisms paradigms, e.g., just for syntactic parsing
  - Shallow and full syntactic parsers
  - Rule-based vs. machine learning based
  - Constituency, Dependency, Combinatory Categorical Grammar, Tree-adjoining grammar and so on
  - Many implementation: Charniak, Stanford, Berkeley,..

- How to combine the different methods in a pipeline to build the desired NLP system?
UIMA

- UIMA supports the development, composition and deployment of multi-modal analytics
  - for the analysis of unstructured information and
  - its integration with search technologies

- Apache UIMA includes
  - APIs and tools for creating analysis components, e.g.
    - tokenizers, summarizers, categorizers, parsers, named-entity detectors etc.
  - Tutorial examples are provided with Apache UIMA
Chapter 2. UIMA Conceptual Overview

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.

The Apache UIMA project is an implementation of the Java UIMA framework available under the Apache License, providing a common foundation for industry and academia to collaborate and accelerate the world-wide development of technologies critical for discovering vital knowledge present in the fastest growing sources of information today.

This chapter presents an introduction to many essential UIMA concepts. It is meant to provide a broad overview to give the reader a quick sense of UIMA's basic architectural philosophy and the UIMA SDK's capabilities.

This chapter provides a general orientation to UIMA and makes liberal reference to the other chapters in the UIMA SDK documentation set, where the reader may find detailed treatments of key concepts and development practices. It may be useful to refer to the Glossary, to become familiar with the terminology in this overview.

2.1. UIMA Introduction

Figure 2.1. UIMA helps you build the bridge between the unstructured and structured worlds

Unstructured information represents the largest, most current and fastest growing source of information available to businesses and governments. The web is just the tip of the iceberg. Consider the mounds of information hosted in the enterprise and around the world and across different media including text, voice and video. The high-value content in these vast collections of unstructured information is, unfortunately, buried in lots of noise. Searching for what you need or doing sophisticated data mining over unstructured information sources presents new challenges.

An unstructured information management (UIM) application may be generally characterized as a software system that analyzes large volumes of unstructured information (text, audio, video, etc.) to identify high-value, most current content that is buried in huge volumes of noise, implicit semantics, and inefficient search.

UIMA: General Purpose IE Pipeline

Analytics bridge the Unstructured & Structured worlds

- Unstructured Information: Text, Chat, Email, Audio, Video
- High-Value, Most Current Content, Buried in Huge Volumes, Lots of Noise, Implicit Semantics, Inefficient Search
- Structured Information: Indices, DBs, KBs

- Identify Semantic Entities, Induce Structure
- Chats, Phone Calls, Transfers
- People, Places, Org, Events
- Times, Topics, Opinions, Relationships
- Threats, Plots, etc.

Explicit Structure, Explicit Semantics, Efficient Search, Focused Content
The Architecture, the Framework and the SDK

- UIMA is a software architecture:
  - component interfaces, data representations, design patterns
  - creates, describes, discovers, composes and deploys multi-modal analysis capabilities

- The **UIMA framework** provides a run-time environment
  - developers can plug in their components
  - these compose UIM applications
The Architecture, the Framework and the SDK

- The framework is not specific to any IDE or platform
  - Apache hosts a Java and (soon) a C++ implementation of the UIMA Framework

- The **UIMA Software Development Kit (SDK)**
  - includes the UIMA framework
  - tools and utilities for using UIMA
  - tools supporting an Eclipse-based (http://www.eclipse.org/) development environment
Analysis Engines, Annotators & Results

- UIMA basic building blocks are called Analysis Engines (AEs)
  - analyze a document and infer and record of descriptive attributes
  - these refer to generally as **analysis results** (metadata)
- Multi-modal analysis: text, audio and video
Primitives of UIMA: begin-end

Analysis Basics

UIMA Version 2.4.0

UIMA Conceptual Overview

The framework is not specific to any IDE or platform. Apache hosts a Java and (soon) a C++ implementation of the UIMA Framework.

The UIMA Software Development Kit (SDK) includes the UIMA framework, plus tools and utilities for using UIMA. Some of the tooling supports an Eclipse-based development environment.

2.3. Analysis Basics

Analysis Engine, Document, Annotator, Annotator Developer, Type, Type System, Feature, Annotation, CAS, Sofa, JCas, UIMA Context.

2.3.1. Analysis Engines, Annotators & Results

Figure 2.2. Objects represented in the Common Analysis Structure (CAS)

UIMA is an architecture in which basic building blocks called Analysis Engines (AEs) are composed to analyze a document and infer and record descriptive attributes about the document as a whole, and/or about regions therein. This descriptive information, produced by AEs is referred to generally as analysis results. Analysis results typically represent meta-data about the document content. One way to think about AEs is as software agents that automatically discover and record meta-data about original content.

UIMA supports the analysis of different modalities including text, audio and video. The majority of examples we provide are for text. We use the term document, therefore, to generally refer to any unit of content that an AE may process, whether it is a text document or a segment of audio, for example. See the section Chapter 6, Multiple CAS Views of an Artifact for more information on multimodal processing in UIMA.

Analysis results include different statements about the content of a document. For example, the following is an assertion about the topic of a document:

(1) The Topic of document D102 is "CEOs and Golf".

Analysis results may include statements describing regions more granular than the entire document. We use the term span to refer to a sequence of characters in a text document. Consider that a document with the identifier D102 contains a span, "Fred Centers" starting at character position 101:

(2) The span from position 101 to 112 in document D102 denotes a Person.

(3) The Person denoted by span 101 to 112 and the Person denoted by span 141 to 143 in document D102 refer to the same Entity.
Primitives of UIMA: Type Annotators

- Basic component types for analysis algorithms running inside AEs
- UIMA framework provides the necessary methods for taking annotators and creating analysis engines
- AEs add the necessary APIs and infrastructure for the composition and deployment of annotators within the UIMA framework.
Representing Analysis Results in the CAS

- Annotators represent and share their results with the **Common Analysis Structure (CAS)**

- The CAS is an object-based data structure:
  - represents objects, properties and values
  - object types may be related to each other in a single-inheritance hierarchy.
  - logically (if not physically) contains the document being analyzed.
  - analytics store results in terms of an object model within the CAS
Example

- For the statement

(2) The span from position 101 to 112 in document D102 denotes a Person

- AE creates a Person object in the CAS and links it to the span of text where the person was mentioned in the document.

- Any type system can be defined in CAS
  - annotation in the document
  - entity as non annotation type
Multiple Views within a CAS

- UIMA supports multiple views of a document
  - for example, the audio and the closed captioned views of a single speech stream
  - the tagged and detagged views of an HTML document
- AEs analyze one or more views of a document, which includes
  - a specific subject of analysis (Sofa)
  - metadata indexed by that view
  - The CAS holds Views and the analysis results
Interacting with the CAS and External Resources

- **Main interfaces:** CAS and the UIMA Context
- **UIMA** provides an efficient implementation of the CAS with multiple programming interfaces
  - read and write analysis results.
  - methods for indexed iterators to the different objects in the CAS, e.g.,
    - a specialized iterator to all Person objects associated with a particular view
jCAS: Java CAS

- JCAS provides a natural interface to CAS objects in Java
  - Each type declared in the type system appears as a Java class, e.g.
  - Person type as a Person class in Java
UIMA Context:

- It’s the framework's resource manager interface
- Allows for accessing external resources
- Can ensure that different annotators working together in an aggregate flow may share the same instance of an external file or remote resource accessed via its URL
Component Descriptors

- Every UIMA component requires:
  1. the declarative part and
  2. the code part

- Component Descriptor is the declarative part
  - contains metadata describing the component, its identity, structure and behavior
  - it is represented in XML

- The code part implements the algorithm, e.g.,
  - a Java program
  - the code may be already provided in reusable subcomponents
Component Descriptors (cont’d)

- Aid in component discovery, reuse, composition and development tooling
- Compose an aggregate engine by pointing to other components
- The UIMA SDK provides tools for easily creating and maintaining the component descriptors
  - relieve the developer from editing XML directly
Component Descriptors (cont’d)

- Contain standard metadata:
  - name, author, version, and a reference to the class that implements the component

- Identify the type system the component uses:
  - the required types from the input CAS
  - and the types it plans to produce in an output CAS

- For example, an AE that detects person types:
  - may require tokenization and deep parse
Component Descriptors (cont’d)

The description refers to a type system:
- input requirements and output types
- a declarative description of the component's behavior
- used in component discovery and composition based on desired results
- UIMA analysis engines provide an interface for accessing the component metadata represented in their descriptors
Aggregate Analysis Engines (AAE)

- A simple AE contains a single annotator
- AEs can contain other AEs organized in a workflow: AAE
- Annotators can be organized in a workflow of component engines and may be orchestrated to perform more complex tasks
An example of AAE

Aggregate Analysis Engine: MyNamedEntityDetector

Language Identifier ➔ Tokenizer ➔ Part of Speech Annotator ➔ Shallow Parser ➔ Named Entity Annotator

CAS Annotations
- Tokens
- Parts of Speech
- Names
- Organizations
- Places
- Persons
Interesting aspects of AAE

- Users of MyNE do not need to know the internal structure
  - only need its name and its published input requirements and output types

- AAE are declared in an AAE descriptors
  - components they contain
  - flow specification: defines the execution order
  - sub AE are called delegate analysis engines
Flow Controller

- Users can define it and include it as part of an aggregate AE by referring to it in the aggregate AE's descriptor.
- Determines the order in which delegate AEs that will process the CAS.
- Can access to the CAS and any external needed resources,
  - dynamically at run-time, it can make multi-step decisions and it can consider any sort of flow specification.
Flow Parallelization

- UIMA framework will run all delegate AEs, ensuring that each one gets access to the CAS in the sequence produced by the flow controller
  - **tightly-coupled** (running in the same process)
  - **loosely-coupled** (running in separate processes or even on different machines).

- UIMA supports a number of remote protocols for loose coupling:
  - SOAP (which stands for Simple Object Access Protocol, a standard Web Services communications protocol)
More on Flow Control

- UIMA can deploy AEs as remote services by using an adapter layer activated by a declaration in the component's descriptor.

- Two built-in flow implementations:
  - a linear flow between components
  - conditional branching based on the document attributes/data

- User-provided flow controllers
  - create multiple AEs and provide their own logic to combine the AEs in arbitrarily complex flows
The UIMA framework, given an aggregate analysis engine descriptor, will run all delegate AEs, ensuring that each one gets access to the CAS in the sequence produced by the flow controller. The UIMA framework is equipped to handle different deployments where the delegate engines, for example, are tightly-coupled (running in the same process) or loosely-coupled (running in separate processes or even on different machines). The framework supports a number of remote protocols for loose coupling deployments of aggregate analysis engines, including SOAP (which stands for Simple Object Access Protocol, a standard Web Services communications protocol). The UIMA framework facilitates the deployment of AEs as remote services by using an adapter layer that automatically creates the necessary infrastructure in response to a declaration in the component's descriptor. For more details on creating aggregate analysis engines refer to Chapter 2, Component Descriptor Reference.

The component descriptor editor tool assists in the specification of aggregate AEs from a repository of available engines. For more details on this tool refer to Chapter 1, Component Descriptor Editor User's Guide.

The UIMA framework implementation has two built-in flow implementations: one that supports a linear flow between components, and one with conditional branching based on the language of the document. It also supports user-provided flow controllers, as described in Chapter 4, Flow Controller Developer's Guide. Furthermore, the application developer is free to create multiple AEs and provide their own logic to combine the AEs in arbitrarily complex flows. For more details on this the reader may refer to Section 3.2, "Using Analysis Engines."
Collection Processing

- Collection Processing Engine (CPE) is an aggregate component
  - specifies a “source to sink” flow from a Collection Reader
  - process it through a set of analysis engines and
    set of CAS Consumers

- Collection Processing Manager reads CPE descriptor, and deploys and runs the specified CPE
Steps of a Collection Processing

1. Connect to a physical source
2. Acquire a document from the source
3. Initialize a CAS with the document to be analyzed
4. Send the CAS to a selected analysis engine
5. Process the resulting CAS
6. Go back to 2 until the collection is processed
7. Do any final processing required after all the documents in the collection have been analyzed
Collection Processing

Graduating to Collection Processing

Figure 2.5. High-Level UIMA Component Architecture from Source to Sink

Many UIM applications analyze entire collections of documents. They connect to different document sources and do different things with the results. But in the typical case, the application must generally follow these logical steps:

1. Connect to a physical source
2. Acquire a document from the source
3. Initialize a CAS with the document to be analyzed
4. Send the CAS to a selected analysis engine
5. Process the resulting CAS
6. Go back to 2 until the collection is processed
7. Do any final processing required after all the documents in the collection have been analyzed

UIMA supports UIM application development for this general type of processing through its Collection Processing Architecture.

As part of the collection processing architecture UIMA introduces two primary components in addition to the annotator and analysis engine. These are the Collection Reader and the CAS Consumer. The complete flow from source, through document analysis, and to CAS Consumers...
A UIMA Collection Processing Engine (CPE) is an aggregate component that specifies a "source to sink" flow from a Collection Reader through a set of analysis engines and then to a set of CAS Consumers. CPEs are specified by XML files called CPE Descriptors. These are declarative specifications that point to their contained components (Collection Readers, analysis engines and CAS Consumers) and indicate a flow among them. The flow specification allows for filtering capabilities, for example, to skip over AEs based on CAS contents. Details about the format of CPE Descriptors may be found in Chapter 3, Collection Processing Engine Descriptor Reference.

The UIMA framework includes a Collection Processing Manager (CPM). The CPM is capable of reading a CPE descriptor, and deploying and running the specified CPE. Figure 2.5, "High-Level UIMA Component Architecture from Source to Sink" illustrates the role of the CPM in the UIMA Framework. Key features of the CPM are failure recovery, CAS management and scale-out.

A Semantic Search engine that works with UIMA is available from IBM's alphaWorks site which will allow the developer to experiment with indexing analysis results and querying for documents based on all the annotations in the CAS. See the section on integrating text analysis and search in Chapter 3, Application Developer's Guide.
Basic Search Engine Implementation

- A Collection Reader reads documents from the file system and initializes CASs with their content
- AE annotates tokens and sentences in the CASs
- CAS Consumer populates a search engine index
- A search engine query processor uses the token index to provide basic key-word search.
Semantic Search Engine

- Supposed to have the AE for NER
- The CAS Consumer will, e.g.,
  - add person and organizations to the CASs by the NER
  - feed these into the semantic search engine's index
- The semantic search engine that is available from http://www.alphaworks.ibm.com/tech/uima supports a query language called **XML Fragments**
Semantic Search Engine (cont’d)

- Queries with meta-data:
  - <organization> center </organization>

- Queries with relations:
  - <ceo_of> <person> center </person> <organization> center </organization> <ceo_of>
Consider taking this one step further. We add a relationship recognizer that annotates mentions of the CEO-of relationship. We configure the CAS Consumer so that it sends these new relationship annotations to the semantic search index as well. With these additional analysis results in the index we can submit queries like:

\[
\text{<ceo_of>}
\text{<person> center </person>}
\text{<organization> center </organization>}
\text{<ceo_of>}
\]

This query will precisely target documents that contain a mention of an organization with "center" as part of its name where that organization is mentioned as part of a CEO-of relationship annotated by the relationship recognizer.

For more details about using UIMA and Semantic Search see the section on integrating text analysis and search in Chapter 3, Application Developer's Guide.

2.6.2. Databases

Search engine indices are not the only place to deposit analysis results for use by applications. Another classic example is populating databases. While many approaches are possible with varying degrees of flexibly and performance all are highly dependent on application specifics. We included a simple sample CAS Consumer that provides the basics for getting your analysis result into a relational database. It extracts annotations from a CAS and writes them to a relational database, using the open source Apache Derby database.

2.7. Multimodal Processing in UIMA

In previous sections we’ve seen how the CAS is initialized with an initial artifact that will be subsequently analyzed by Analysis engines and CAS Consumers. The first Analysis engine may make some assertions about the artifact, for example, in the form of annotations. Subsequent Analysis engines will make further assertions about both the artifact and previous analysis results, and finally one or more CAS Consumers will extract information from these CASs for structured information storage.

- Several Sofas associated with multiple CAS views
- Components written in multiple-view mode
  - analyze CAS according to different Sofas