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:
  - Tokenizers, Sentence Splitter, Topic Categorization, Pos-Tagging, Syntactic Parsing, Shallow Semantic Parsing, Coreference Resolution, Relation Extraction, Textual Entailment, Semantic Role Labeling, Opinion Miners, Disambiguation Module, Named Entity Recognition and Normalization…

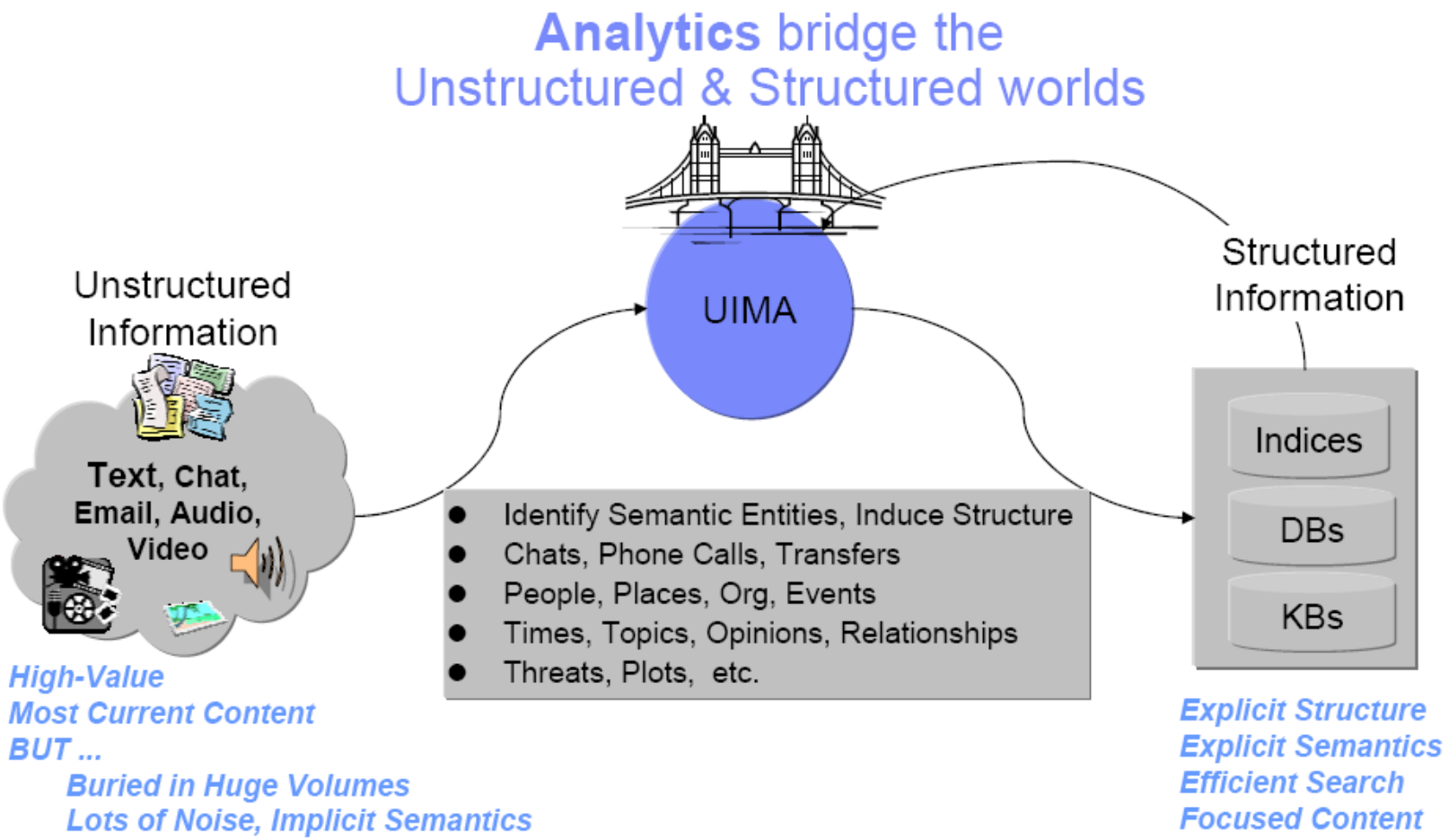
Motivations

- 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

UIMA: General Purpose IE Pipeline



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

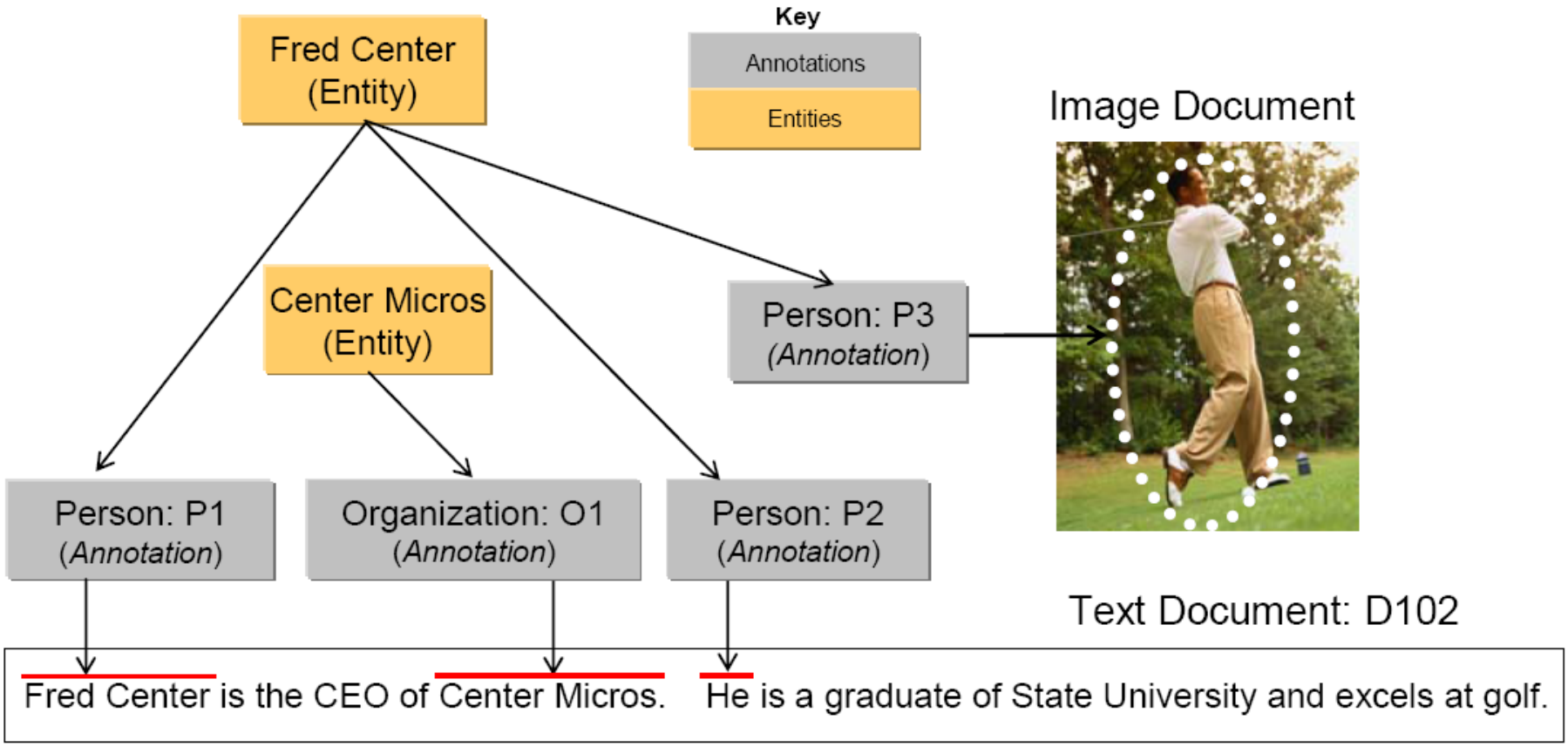
The Architecture, the Framework and the SDK

- 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** (meta-data)
- Multi-modal analysis: text, audio and video

Primitives of UIMA: begin-end



(2) The head form of document D102 is 'CEO' and 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

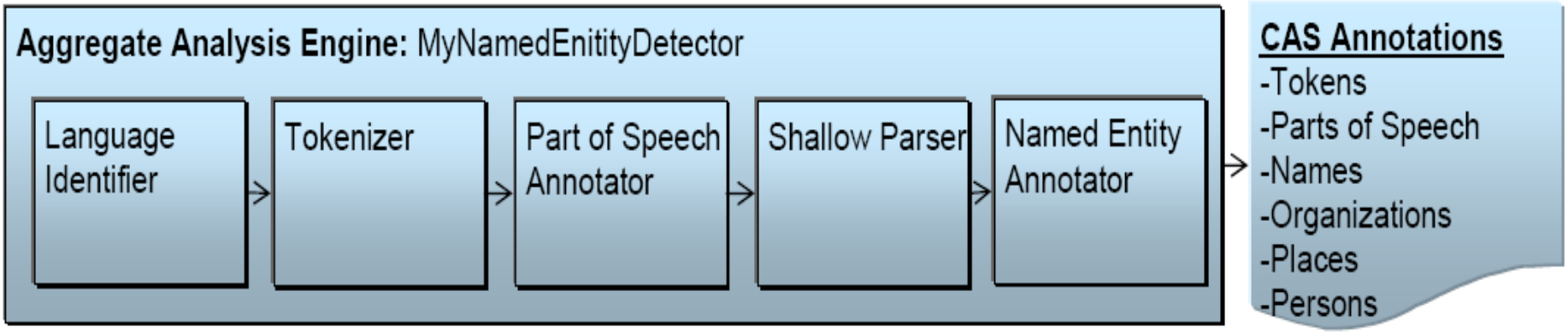
Component Descriptors (cont'd)

- 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



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

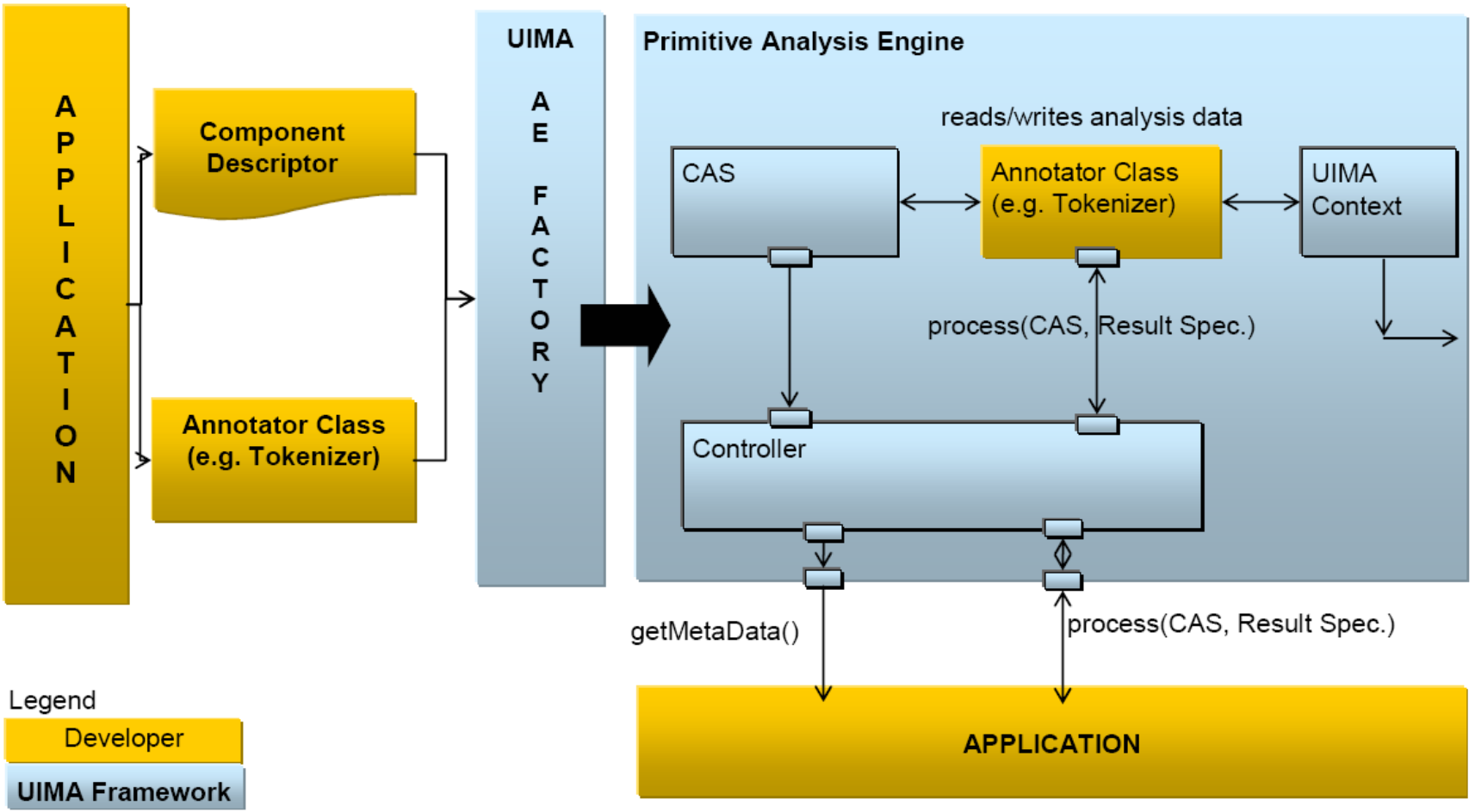
Flow Parallelization

- 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

Example of Interaction with an analysis engine



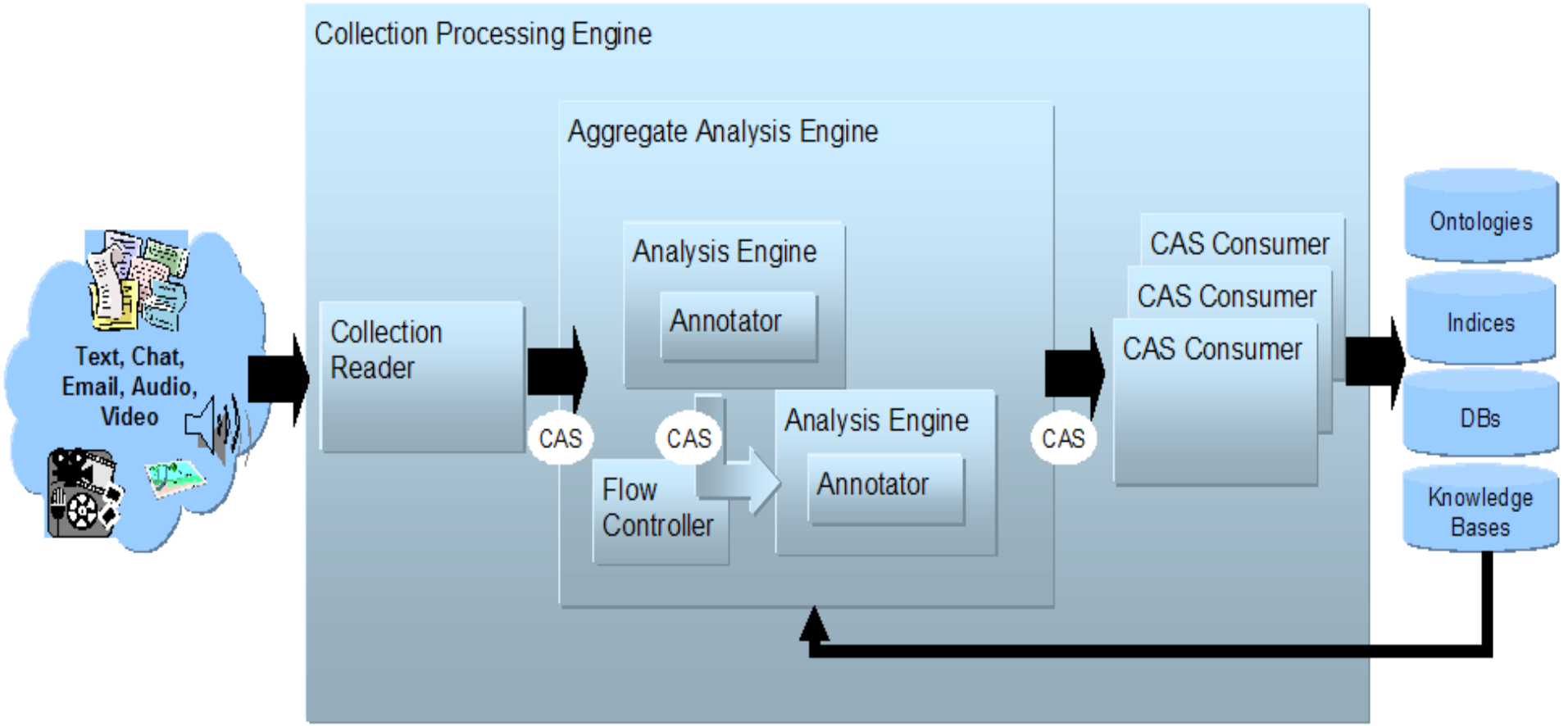
Collection Processing

- **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

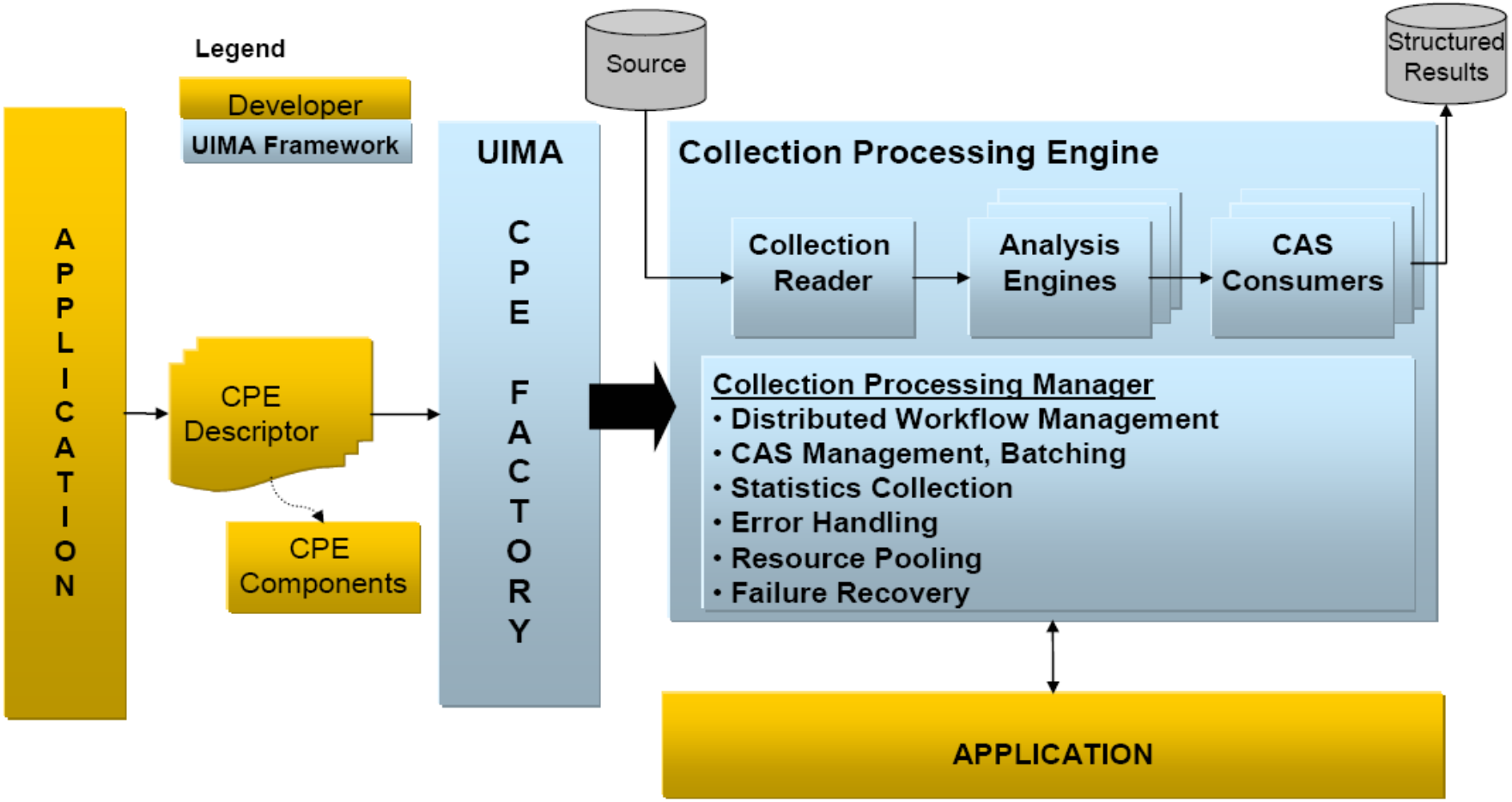
Steps of a Collection Processing

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



Collection Processing Engine



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 use the token index to provide basic key-word search.

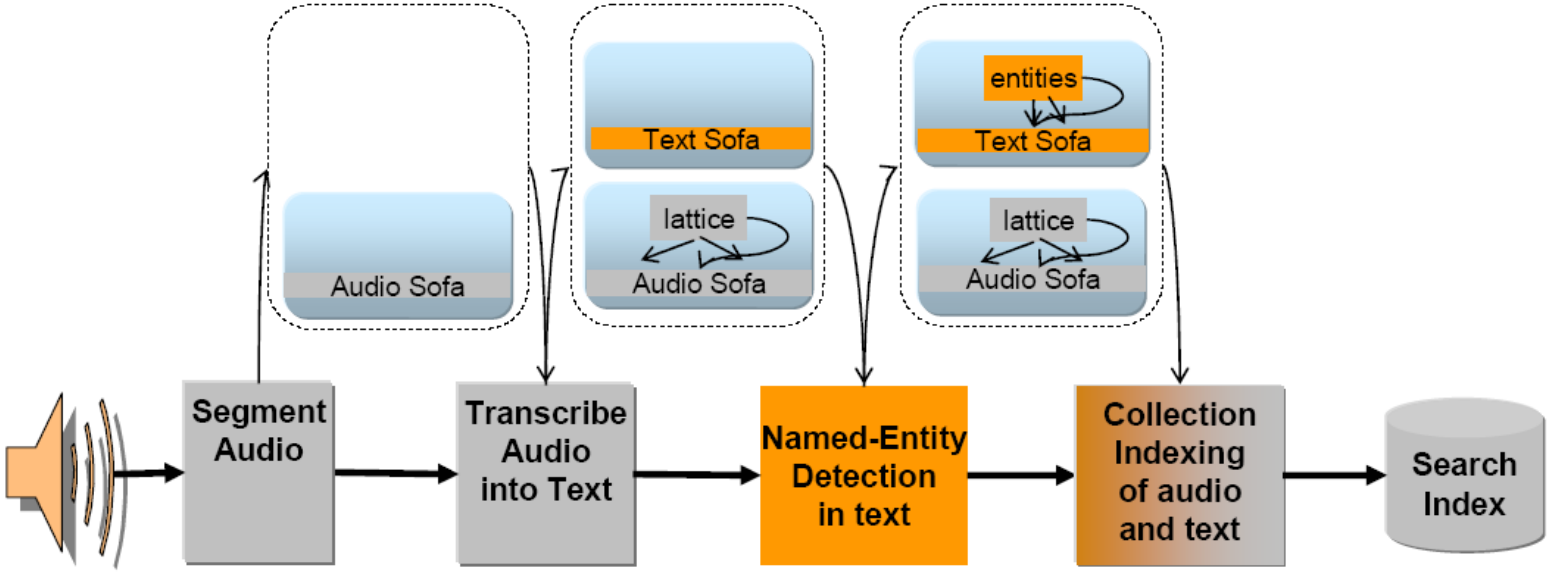
Semantic Search Engine

- 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>

Multimodal Processing in UIMA



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