Einführung in Apache UIMA

Dr. Christian Herta

June 22, 2009

Outline

Formen von Information

- Recap Unterscheidung: Daten, Information, Wissen

Formen von Information

- Recap Unterscheidung: Daten, Information, Wissen
- Strukturierte Information: Datenbanken, Ontologien, XML-annotierter Text

Formen von Information

- Recap Unterscheidung: Daten, Information, Wissen
- Strukturierte Information: Datenbanken, Ontologien, XML-annotierter Text
- Semi-strukturierte Information: Text mit Darstellungsmarkups (Web-Seiten), Dokumente in XML-Formate (z.B. Trennung: Überschrift, Zusammenfassung, Schlagworte, Haupttext)

Formen von Information

- Recap Unterscheidung: Daten, Information, Wissen
- Strukturierte Information: Datenbanken, Ontologien, XML-annotierter Text
- Semi-strukturierte Information: Text mit Darstellungsmarkups (Web-Seiten), Dokumente in XML-Formate (z.B. Trennung: Überschrift, Zusammenfassung, Schlagworte, Haupttext)
- Unstrukturierte Information: Text, Audio, Video

Unstructured Information Management

- Mehrheit der Daten liegt in unstrukturierter Form, wie Text-Dokumenten, vor (vgl. Einführungsvorlesung)

Unstructured Information Management

- Mehrheit der Daten liegt in unstrukturierter Form, wie Text-Dokumenten, vor (vgl. Einführungsvorlesung)
- Automatische Anreicherung mit Meta-Daten kann Anwendungen (zur Nutzung der Daten) erheblich verbessern, z.B. für Informationsstyteme, Wissensmanagement und Suche

Unstructured Information Management

- Mehrheit der Daten liegt in unstrukturierter Form, wie Text-Dokumenten, vor (vgl. Einführungsvorlesung)
- Automatische Anreicherung mit Meta-Daten kann Anwendungen (zur Nutzung der Daten) erheblich verbessern, z.B. für Informationsstyteme, Wissensmanagement und Suche
- Dies wird als *Unstructured Information Management (UIM)* bezeichnet.

Unstructured Information Management

- Mehrheit der Daten liegt in unstrukturierter Form, wie Text-Dokumenten, vor (vgl. Einführungsvorlesung)
- Automatische Anreicherung mit Meta-Daten kann Anwendungen (zur Nutzung der Daten) erheblich verbessern, z.B. für Informationsstyteme, Wissensmanagement und Suche
- Dies wird als *Unstructured Information Management (UIM)* bezeichnet.
- Dabei werden meist strukturierte Daten aus unstrukturierten Daten gewonnen.

Technologien die *Unstructured Information Management* unterstützen

- Statistical and rule-based Natural Language Processing

Technologien die *Unstructured Information Management* unterstützen

- Statistical and rule-based Natural Language Processing
- Information Retrieval

Technologien die *Unstructured Information Management* unterstützen

- Statistical and rule-based Natural Language Processing
- Information Retrieval
- Machine Learning

Technologien die *Unstructured Information Management* unterstützen

- Statistical and rule-based Natural Language Processing
- Information Retrieval
- Machine Learning
- Ontologies

Technologien die *Unstructured Information Management* unterstützen

- Statistical and rule-based Natural Language Processing
- Information Retrieval
- Machine Learning
- Ontologies
- Automated Resoning

Technologien die *Unstructured Information Management* unterstützen

- Statistical and rule-based Natural Language Processing
- Information Retrieval
- Machine Learning
- Ontologies
- Automated Resoning
- Knowledge Sources (CYC, DBpedia, WordNet, FrameNet, Geonames etc.)

UIMA - Keyfacts

- UIMA (*you-eee-muh*) steht für *Unstructured Information Management Architecture*: Architektur zur Verwaltung unstrukturierter Informationen

UIMA - Keyfacts

- UIMA (*you-eee-muh*) steht für *Unstructured Information Management Architecture*: Architektur zur Verwaltung unstrukturierter Informationen
- UIMA ist eine Architektur und ein Framework zu Erzeugung von UIM-Anwendungen.

UIMA - Keyfacts

- UIMA (*you-eee-muh*) steht für *Unstructured Information Management Architecture*: Architektur zur Verwaltung unstrukturierter Informationen
- UIMA ist eine Architektur und ein Framework zu Erzeugung von UIM-Anwendungen.
- Apache Incubator Projekt; ursprünglich von IBM entwickelt und als Open Source freigegeben

UIMA - Keyfacts

- UIMA (*you-eee-muh*) steht für *Unstructured Information Management Architecture*: Architektur zur Verwaltung unstrukturierter Informationen
- UIMA ist eine Architektur und ein Framework zu Erzeugung von UIM-Anwendungen.
- Apache Incubator Projekt; ursprünglich von IBM entwickelt und als Open Source freigegeben
- Lizenz: *Apache Licence*

UIMA - Keyfacts

- UIMA (*you-eee-muh*) steht für *Unstructured Information Management Architecture*: Architektur zur Verwaltung unstrukturierter Informationen
- UIMA ist eine Architektur und ein Framework zu Erzeugung von UIM-Anwendungen.
- Apache Incubator Projekt; ursprünglich von IBM entwickelt und als Open Source freigegeben
- Lizenz: *Apache Licence*
- UIMA ist seit März 2009 OASIS Standard

UIMA - Keyfacts

- UIMA (*you-eee-muh*) steht für *Unstructured Information Management Architecture*: Architektur zur Verwaltung unstrukturierter Informationen
- UIMA ist eine Architektur und ein Framework zu Erzeugung von UIM-Anwendungen.
- Apache Incubator Projekt; ursprünglich von IBM entwickelt und als Open Source freigegeben
- Lizenz: *Apache Licence*
- UIMA ist seit März 2009 OASIS Standard

Zweck von UIMA nach [1]

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

Ziele und Vorteile von UIMA

Ziele und Vorteile von UIMA

- Standardisierte offene Plattform

Ziele und Vorteile von UIMA

- Standardisierte offene Plattform
- Wiederverwendung von NLP-Komponenten

Ziele und Vorteile von UIMA

- Standardisierte offene Plattform
- Wiederverwendung von NLP-Komponenten
- performant, skalierbar durch Parallelisierbarkeit und Verteilung

Ziele und Vorteile von UIMA

- Standardisierte offene Plattform
- Wiederverwendung von NLP-Komponenten
- performant, skalierbar durch Parallelisierbarkeit und Verteilung
- Verschiedene Programmiersprachen: Java, C++; (auch Perl, Python, TCL)

Modularitätsprinzip

Modularitätsprinzip

- Eigenständige wiederverwertbare Komponenten für Teilaufgaben, wie

Modularitätsprinzip

- Eigenständige wiederverwertbare Komponenten für Teilaufgaben, wie
  - Konvertierung des Formates (z.B. *HTML Stripping*)

Modularitätsprinzip

- Eigenständige wiederverwertbare Komponenten für Teilaufgaben, wie
  - Konvertierung des Formates (z.B. *HTML Stripping*)
  - Klassifikation

Modularitätsprinzip

- Eigenständige wiederverwertbare Komponenten für Teilaufgaben, wie
  - Konvertierung des Formates (z.B. *HTML Stripping*)
  - Klassifikation
  - *Denoising* von Web-Dokumenten

Modularitätsprinzip

- Eigenständige wiederverwertbare Komponenten für Teilaufgaben, wie
  - Konvertierung des Formates (z.B. *HTML Stripping*)
  - Klassifikation
  - *Denoising* von Web-Dokumenten
  - Tokenizer

Modularitätsprinzip

- Eigenständige wiederverwertbare Komponenten für Teilaufgaben, wie
  - Konvertierung des Formates (z.B. *HTML Stripping*)
  - Klassifikation
  - *Denoising* von Web-Dokumenten
  - Tokenizer
  - *Sentencesplitter*

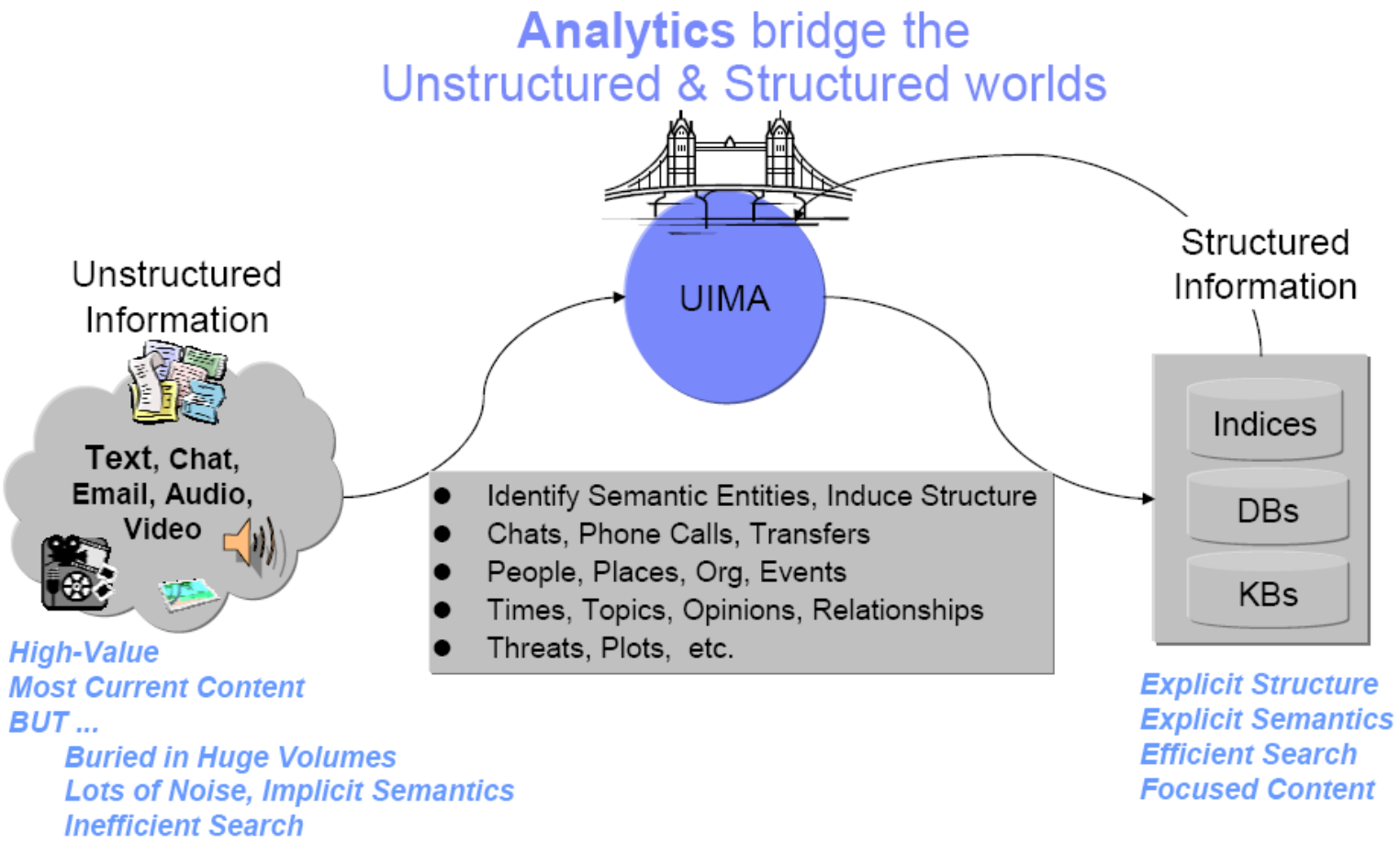
Modularitätsprinzip

- Eigenständige wiederverwertbare Komponenten für Teilaufgaben, wie
  - Konvertierung des Formates (z.B. *HTML Stripping*)
  - Klassifikation
  - *Denoising* von Web-Dokumenten
  - Tokenizer
  - *Sentencesplitter*
  - POS-Tagger

Modularitätsprinzip

- Eigenständige wiederverwertbare Komponenten für Teilaufgaben, wie
  - Konvertierung des Formates (z.B. *HTML Stripping*)
  - Klassifikation
  - *Denoising* von Web-Dokumenten
  - Tokenizer
  - *Sentencesplitter*
  - POS-Tagger
  - Named Entity Recognition

UIMA als Brücke zwischen unstrukturierten und strukturierten Daten



Beispiel für Annotationen

Erklärung siehe Tafel

Zusammenspiel der Verarbeitungskomponenten

Zusammenspiel der Verarbeitungskomponenten

- Zusammenschalten von Komponenten - Datenaustausch mittels gemeinsamen, kompatiblen Austauschformat

Zusammenspiel der Verarbeitungskomponenten

- Zusammenschalten von Komponenten - Datenaustausch mittels gemeinsamen, kompatiblen Austauschformat
- Hintereinanderschaltung der Komponenten zu Verarbeitungspipelines

Zusammenspiel der Verarbeitungskomponenten

- Zusammenschalten von Komponenten - Datenaustausch mittels gemeinsamen, kompatiblen Austauschformat
- Hintereinanderschaltung der Komponenten zu Verarbeitungspipelines
- Mischen von Komponenten verschiedener Programmiersprachen innerhalb einer Verarbeitungspipline möglich

Zusammenspiel der Verarbeitungskomponenten

- Zusammenschalten von Komponenten - Datenaustausch mittels gemeinsamen, kompatiblen Austauschformat
- Hintereinanderschaltung der Komponenten zu Verarbeitungspipelines
- Mischen von Komponenten verschiedener Programmiersprachen innerhalb einer Verarbeitungspipline möglich
- Verarbeitungskomponenten werden in einem XML-File beschrieben - unterstützt von Eclipse Plugins

Zusammenspiel der Verarbeitungskomponenten

- Zusammenschalten von Komponenten - Datenaustausch mittels gemeinsamen, kompatiblen Austauschformat
- Hintereinanderschaltung der Komponenten zu Verarbeitungspipelines
- Mischen von Komponenten verschiedener Programmiersprachen innerhalb einer Verarbeitungspipline möglich
- Verarbeitungskomponenten werden in einem XML-File beschrieben - unterstützt von Eclipse Plugins
- Wiederverwendbarkeit und Integration unterschiedlicher Tools möglich

Outline

Übersicht: wichtige UIMA Module

- UIMA Framework Core
- CAS (*Common Analysis Structur*)
- AE (*Analysis Engines*)
- CPE (*Collection Processing Engine*)
- CPM (*Collection Processing Management*)

UIMA Framework Core

- Basis Infrastruktur und SDK
- Stellt die Grundfunktionalität bereit
- Apache UIMA Java Framework: Java-basierte Implementation der UIMA Architektur: Stellt eine *run-time environment* bereit, mit der Entwickler ihre Komponenten entwicklen, zusammenschalten und *deployen* können.

Analysis Engine (AE)

Analysis Engines

Verarbeitungskomponenten heißen Analysis Engines (AEs)

Analysis Engine (AE)

Analysis Engines

Verarbeitungskomponenten heißen Analysis Engines (AEs)

- *Basic Building Blocks* der Analyse

Analysis Engine (AE)

Analysis Engines

Verarbeitungskomponenten heißen Analysis Engines (AEs)

- *Basic Building Blocks* der Analyse
- Erzeugen (in der Regel) von Meta-Daten zum Dokument-Inhalt

Analysis Engine (AE)

Analysis Engines

Verarbeitungskomponenten heißen Analysis Engines (AEs)

- *Basic Building Blocks* der Analyse
- Erzeugen (in der Regel) von Meta-Daten zum Dokument-Inhalt
- Verarbeitende Artefakte (Dokumente), sind nicht nur Text-Dokumente sondern auch Audio, Video, Bilder etc.

Analysis Engine (AE)

Analysis Engines

Verarbeitungskomponenten heißen Analysis Engines (AEs)

- *Basic Building Blocks* der Analyse
- Erzeugen (in der Regel) von Meta-Daten zum Dokument-Inhalt
- Verarbeitende Artefakte (Dokumente), sind nicht nur Text-Dokumente sondern auch Audio, Video, Bilder etc.
- Text Analysis Engine (TAE) sind AEs, die auf Text operieren

Analysis Engine (AE)

Analysis Engines

Verarbeitungskomponenten heißen Analysis Engines (AEs)

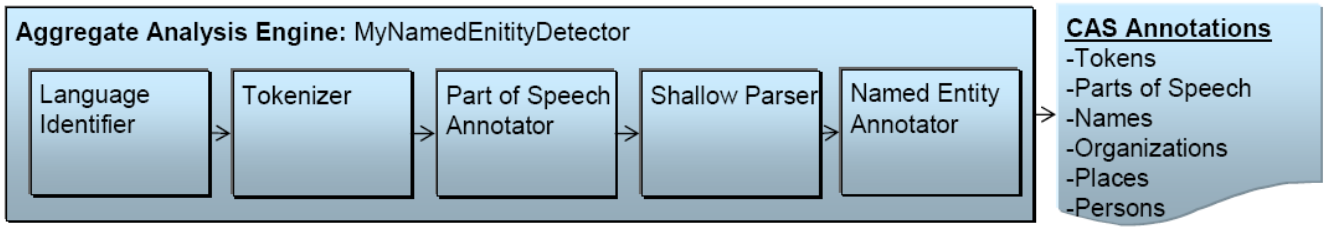
- *Basic Building Blocks* der Analyse
- Erzeugen (in der Regel) von Meta-Daten zum Dokument-Inhalt
- Verarbeitende Artefakte (Dokumente), sind nicht nur Text-Dokumente sondern auch Audio, Video, Bilder etc.
- Text Analysis Engine (TAE) sind AEs, die auf Text operieren

Definition: *Analysis Result*

Das Ergebnis der Analyse (eines AEs) wird (*Analysis Result*) genannt.

Typen von Analysis Engine (AE)

- zwei Typen von AEs
- *Primitive AEs*: Einzel AEs
- *Aggregate AEs*: mehrere AEs, die zu einem neuen *Aggregate AE* zusammengeschaltet sind.



CAS: Common Analysis Structure

CAS - *Common Analysis Structure*

Die Ergebnisse der AEs (*Analysis results*) werden in einer speziellen Datenstruktur repräsentiert. Diese wird CAS (*Common Analysis Structure*) genannt.

CAS: Common Analysis Structure

CAS - *Common Analysis Structure*

Die Ergebnisse der AEs (*Analysis results*) werden in einer speziellen Datenstruktur repräsentiert. Diese wird CAS (*Common Analysis Structure*) genannt.

- CAS besteht aus:
  - Artefakt (Dokument)

CAS: Common Analysis Structure

CAS - *Common Analysis Structure*

Die Ergebnisse der AEs (*Analysis results*) werden in einer speziellen Datenstruktur repräsentiert. Diese wird CAS (*Common Analysis Structure*) genannt.

- CAS besteht aus:
  - Artefakt (Dokument)
  - Typsystem-Beschreibung

CAS: Common Analysis Structure

CAS - *Common Analysis Structure*

Die Ergebnisse der AEs (*Analysis results*) werden in einer speziellen Datenstruktur repräsentiert. Diese wird CAS (*Common Analysis Structure*) genannt.

- CAS besteht aus:
  - Artefakt (Dokument)
  - Typsystem-Beschreibung
  - Metadaten der Analyse

CAS: Common Analysis Structure

CAS - *Common Analysis Structure*

Die Ergebnisse der AEs (*Analysis results*) werden in einer speziellen Datenstruktur repräsentiert. Diese wird CAS (*Common Analysis Structure*) genannt.

- CAS besteht aus:
  - Artefakt (Dokument)
  - Typsystem-Beschreibung
  - Metadaten der Analyse
  - Indizes (zum Zugriff auf die (*Analysis results*)

CAS: Common Analysis Structure

CAS - *Common Analysis Structure*

Die Ergebnisse der AEs (*Analysis results*) werden in einer speziellen Datenstruktur repräsentiert. Diese wird CAS (*Common Analysis Structure*) genannt.

- CAS besteht aus:
  - Artefakt (Dokument)
  - Typsystem-Beschreibung
  - Metadaten der Analyse
  - Indizes (zum Zugriff auf die (*Analysis results*)
- CAS ist die Datenstruktur, die zwischen den AEs ausgetauscht wird

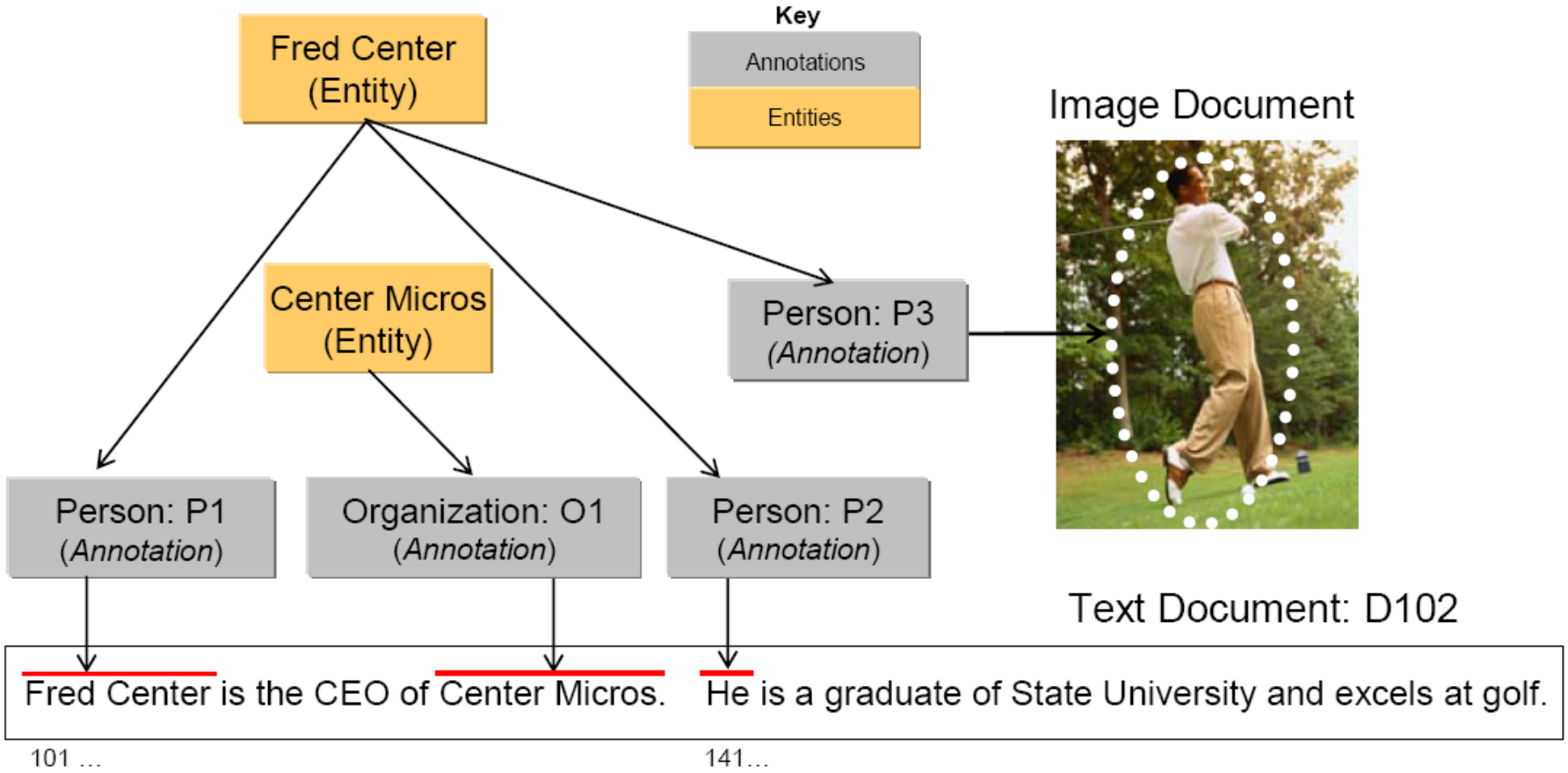
Typsystem

- Im Typsystem (*type system*) wird spezifiziert, welche Daten in den AEs manipuliert (und erzeugt) werden können

Typsystem

- Im Typsystem (*type system*) wird spezifiziert, welche Daten in den AEs manipuliert (und erzeugt) werden können
- Das Typsystem besteht aus:
  - *Types* (analog: Java-Klassen ohne Methoden)
  - *Features* (analog: Attribute der Klassen)
- Feature Structures (analog: Instanzen der Java-Klassen(Types))
- Note: UIMA Typsystem unterstützt auch die Erweiterung von Typen (kein Java Analogon)

Beispiel für ein *type system*



Types

- Organisiert in Einfachvererbungshierarchie

Types

- Organisiert in Einfachvererbungshierarchie
- Wurzel Oberklasse ist `uima.cas.TOP` (ohne *Features*)
- Java Klasse: `org.apache.uima.jcas.cas.TOP`

Types

- Organisiert in Einfachvererbungshierarchie
- Wurzel Oberklasse ist `uima.cas.TOP` (ohne *Features*)
- Java Klasse: `org.apache.uima.jcas.cas.TOP`
- Unterklassen erben alle *Features* der Oberklassen

Annotationen

- Annotationen sind ein Spezialfall der *Types*
- Java Klasse: `org.apache.uima.jcas.tcas.Annotation`
- zusätzliche Features von Annoations
  - Start-Offset
  - End-Offset

Wichtige Methoden von *Annotationen*

- *setter* und *getter* Funktionen für die Start- und Endposition der Annotationen

Wichtige Methoden von *Annotationen*

- *setter* und *getter* Funktionen für die Start- und Endposition der Annotationen
- `String getCoveredText()`:Get the text covered by an annotation as a string.

Collection Processing

- Bisher: Komponenten zur Verarbeitung eines Dokuments

Collection Processing

- Bisher: Komponenten zur Verarbeitung eines Dokuments
- Offen wie mit Dokument-Collections umgegangen wird

Collection Processing

- Bisher: Komponenten zur Verarbeitung eines Dokuments
- Offen wie mit Dokument-Collections umgegangen wird
- *Collection Processing Architecture* - Aufgaben

Collection Processing

- Bisher: Komponenten zur Verarbeitung eines Dokuments
- Offen wie mit Dokument-Collections umgegangen wird
- *Collection Processing Architecture* - Aufgaben
  - Lesen von *raw data formats* aus *data collections*

Collection Processing

- Bisher: Komponenten zur Verarbeitung eines Dokuments
- Offen wie mit Dokument-Collections umgegangen wird
- *Collection Processing Architecture* - Aufgaben
  - Lesen von *raw data formats* aus *data collections*
  - Präparieren der Daten (Erzeugung von Einzeldokumente für die Verarbeitung

Collection Processing

- Bisher: Komponenten zur Verarbeitung eines Dokuments
- Offen wie mit Dokument-Collections umgegangen wird
- *Collection Processing Architecture* - Aufgaben
  - Lesen von *raw data formats* aus *data collections*
  - Präparieren der Daten (Erzeugung von Einzeldokumente für die Verarbeitung
  - Steuerung und Ausführen der Analyse

Collection Processing

- Bisher: Komponenten zur Verarbeitung eines Dokuments
- Offen wie mit Dokument-Collections umgegangen wird
- *Collection Processing Architecture* - Aufgaben
  - Lesen von *raw data formats* aus *data collections*
  - Präparieren der Daten (Erzeugung von Einzeldokumente für die Verarbeitung
  - Steuerung und Ausführen der Analyse
  - *Deployment* des *flows*: lokal und verteilt

Collection Processing

- Bisher: Komponenten zur Verarbeitung eines Dokuments
- Offen wie mit Dokument-Collections umgegangen wird
- *Collection Processing Architecture* - Aufgaben
  - Lesen von *raw data formats* aus *data collections*
  - Präparieren der Daten (Erzeugung von Einzeldokumente für die Verarbeitung
  - Steuerung und Ausführen der Analyse
  - *Deployment* des *flows*: lokal und verteilt

Collection Processing Engine - CPE

Die Funktionalität des *Collection Processing* wird von *CPEs* umgesetzt

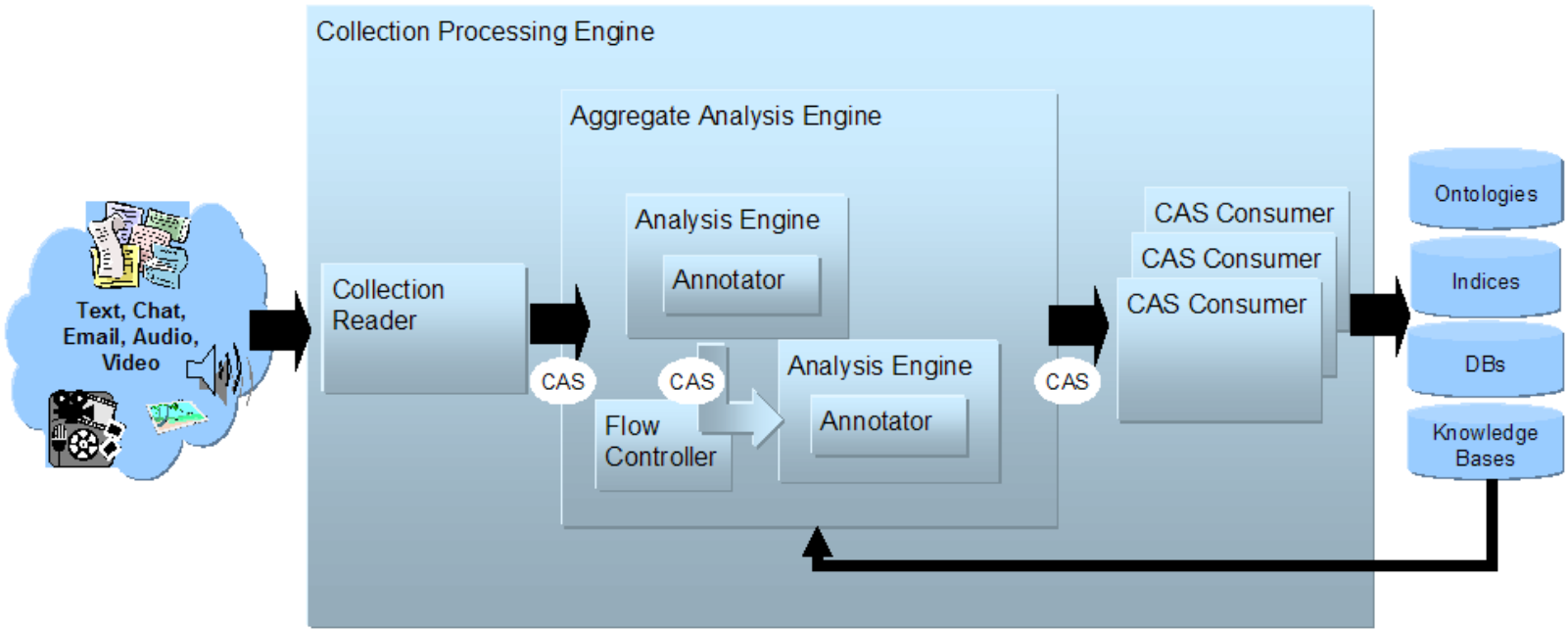
Komponenten einer *Collection Processing Engines*

- CPEs werden deklarativ in einer XML-Datei beschrieben oder mittels eines graphische Tools.
- Komponenten einer CPE werden deklariert:
  - *Collection Reader*: Schnittstelle zum Einlesen von Daten (Dokumenten); gibt CASes zu Analyse weiter
  - *CAS Processors*

    Analysis Engines (*AEs*): Analyse des Inhaltes und Füllen/Manipulation der CASes
    - *CAS Consumer*: Konsumieren CASes - Schreiben den CAS-Inhalt in der Regel in Datenbanken, Dateien oder Indizes

      In Zukunft sollen AEs die Rolle der *CAS Consumer* übernehmen

Zusammenspiel der Module mittels CPE



Einbinden von CPEs in eigenen Anwendungscode

Listing 1: Einbinden von CPEs in eigenen Anwendungscode

```
1  //parse CPE descriptor in file specified on command line
2  CpeDescription cpeDesc = UIMAFramework.getXMLParser().
3          parseCpeDescription(new XMLInputSource(args[0]));
4
5          //instantiate CPE
6  mCPE = UIMAFramework.produceCollectionProcessingEngine(cpeDesc);
7
8          //Create and register a Status Callback Listener
9  mCPE.addStatusCallbackListener(new StatusCallbackListenerImpl());
10
11         //Start Processing
12 mCPE.process();
```

Collection Processing Manager

Collection Processing Manager (CPM)

CPMs managen die Ausführung von CPEs

Collection Processing Manager

Collection Processing Manager (CPM)

CPMs managen die Ausführung von CPEs

Aufgaben der CPM

- Lesen der CPE-Spezifikation und instanziieren und starten einer CPE Instanz

Collection Processing Manager

Collection Processing Manager (CPM)

CPMs managen die Ausführung von CPEs

Aufgaben der CPM

- Lesen der CPE-Spezifikation und instanziieren und starten einer CPE Instanz
- Orchestrierung des Datenfluss innerhalb einer CPE

Collection Processing Manager

Collection Processing Manager (CPM)

CPMs managen die Ausführung von CPEs

Aufgaben der CPM

- Lesen der CPE-Spezifikation und instanziieren und starten einer CPE Instanz
- Orchestrierung des Datenfluss innerhalb einer CPE
- Monitoring des Status

Collection Processing Manager

Collection Processing Manager (CPM)

CPMs managen die Ausführung von CPEs

Aufgaben der CPM

- Lesen der CPE-Spezifikation und instanziieren und starten einer CPE Instanz
- Orchestrierung des Datenfluss innerhalb einer CPE
- Monitoring des Status
- Managing des *life-cycle* von internen Komponenten

Collection Processing Manager

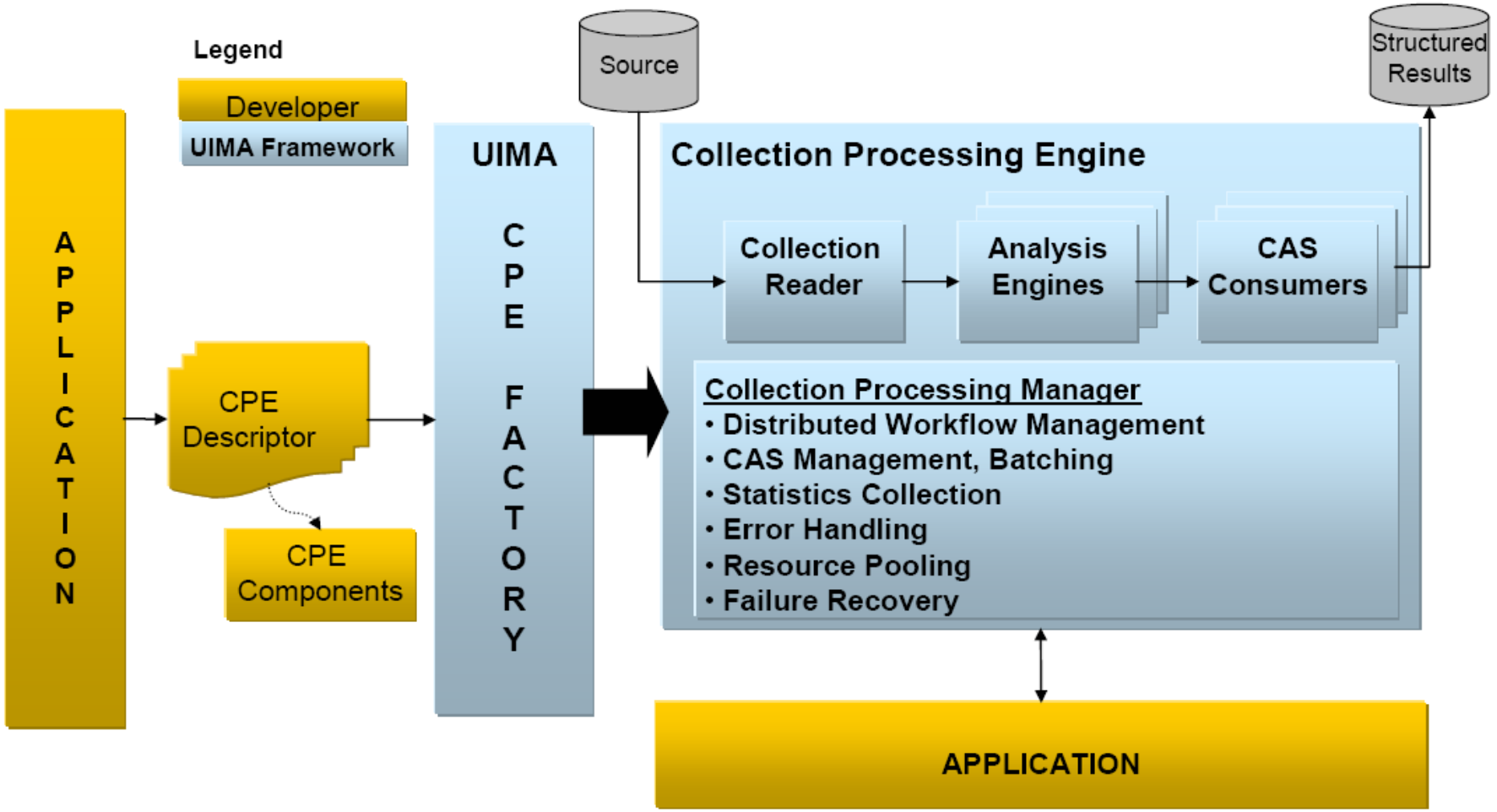
Collection Processing Manager (CPM)

CPMs managen die Ausführung von CPEs

Aufgaben der CPM

- Lesen der CPE-Spezifikation und instanziieren und starten einer CPE Instanz
- Orchestrierung des Datenfluss innerhalb einer CPE
- Monitoring des Status
- Managing des *life-cycle* von internen Komponenten
- Erstellen von *Collection* Statistik

Aufgabe des CPM



Outline

Views und Sofas

- (Gleichzeitige) Analyse von verschiedenen Sichten (*views*) auf ein Artefakt (und assozierten Types) ist möglich
- CAS Views
- Sofa *Subject-of-Analysis*: Dokument, FS und URI
- Eins-zu-ein Korrespondenz zwischen View und Sofa: jedes CAS View hat ein assoziertes Sofa
- Beispiel für zwei Sichten (*views*):
  - HTML-Sicht und Textsicht eines Web-Dokumentes

Views und Sofas

- Views und assoziierte Sofas haben **einen** Namen
- Der Name wird wird aus historischen Gründen *sofa name* genannt
- Namen werden deklariert in den XML-Matadaten der Komponente

Outline

Entwicklung von *Collection Reader*

Entwicklung von *Collection Reader*

- Collection Reader müssen die Schnittstelle `org.apache.uima.collection.CollectionReader` implementieren

Entwicklung von *Collection Reader*

- Collection Reader müssen die Schnittstelle `org.apache.uima.collection.CollectionReader` implementieren
- einfacher: Erweiterung der *Convenience base class* `org.apache.uima.collection.CollectionReader_ImplBase`

Entwicklung von *Collection Reader*

- Collection Reader müssen die Schnittstelle `org.apache.uima.collection.CollectionReader` implementieren
- einfacher: Erweiterung der *Convenience base class* `org.apache.uima.collection.CollectionReader_ImplBase`
- Implementierung der abstrakten Methoden nötig
  - `initialize()` (nicht abstrakt): Resourcen initialisieren, Zugriff auf Parameter

Entwicklung von *Collection Reader*

- Collection Reader müssen die Schnittstelle `org.apache.uima.collection.CollectionReader` implementieren
- einfacher: Erweiterung der *Convenience base class* `org.apache.uima.collection.CollectionReader_ImplBase`
- Implementierung der abstrakten Methoden nötig
  - `initialize()`(nicht abstrakt): Resourcen initialisieren, Zugriff auf Parameter
  - `hasNext()`: Rückgabe, ob noch Dokumente zum Einlesen vorhanden sind

Entwicklung von *Collection Reader*

- Collection Reader müssen die Schnittstelle `org.apache.uima.collection.CollectionReader` implementieren
- einfacher: Erweiterung der *Convenience base class* `org.apache.uima.collection.CollectionReader_ImplBase`
- Implementierung der abstrakten Methoden nötig
  - `initialize()` (nicht abstrakt): Resourcen initialisieren, Zugriff auf Parameter
  - `hasNext()`: Rückgabe, ob noch Dokumente zum Einlesen vorhanden sind
  - `getNext(CAS)`: Einlesen des nächsten Dokumentes und Populierung eines CAS

Entwicklung von *Collection Reader*

- Collection Reader müssen die Schnittstelle `org.apache.uima.collection.CollectionReader` implementieren
- einfacher: Erweiterung der *Convenience base class* `org.apache.uima.collection.CollectionReader_ImplBase`
- Implementierung der abstrakten Methoden nötig
  - `initialize()`(nicht abstrakt): Resourcen initialisieren, Zugriff auf Parameter
  - `hasNext()`: Rückgabe, ob noch Dokumente zum Einlesen vorhanden sind
  - `getNext(CAS)`: Einlesen des nächsten Dokumentes und Populierung eines CAS
  - `getProgress()`: Fortschritt-Status Abfrage

Entwicklung von *Collection Reader*

- Collection Reader müssen die Schnittstelle `org.apache.uima.collection.CollectionReader` implementieren
- einfacher: Erweiterung der *Convenience base class* `org.apache.uima.collection.CollectionReader_ImplBase`
- Implementierung der abstrakten Methoden nötig
  - `initialize()`(nicht abstrakt): Resourcen initialisieren, Zugriff auf Parameter
  - `hasNext()`: Rückgabe, ob noch Dokumente zum Einlesen vorhanden sind
  - `getNext(CAS)`: Einlesen des nächsten Dokumentes und Populierung eines CAS
  - `getProgress()`: Fortschritt-Status Abfrage
  - `close()`: Freigeben benutzter Resourcen

Beispielcode zum Setzen von CAS Content

Listing 2: Beispielcode zum Setzen von CAS Content

```
1  public void getNext(CAS aCas){
2      //...
3      try{
4          JCas jCas = aCas. getJCas();
5          jcas.setDocumentText(documentText);
6          DocumentMetadata metadata = new DocumentMetadata(jcas);
7          metadata.setDocumentURL(docURL);
8          //set more metadata content
9          jcas.addFsToIndexes(metadata);
10     } catch (CASException e) {
11         throw new CollectionException(e);
12     }
13     // ...
14 }
```

Entwicklung von *CAS Consumern*

- Note: Ab UIMA 2.x sind CAS Consumer spezielle AEs - In Zukunft sollte man statt CAS Consumer AEs verwenden

Entwicklung von *CAS Consumern*

- Note: Ab UIMA 2.x sind CAS Consumer spezielle AEs - In Zukunft sollte man statt CAS Consumer AEs verwenden
- CAS Consumer müssen die Schnittstelle `org.apache.uima.collection.CasConsumer` implementieren

Entwicklung von *CAS Consumern*

- Note: Ab UIMA 2.x sind CAS Consumer spezielle AEs - In Zukunft sollte man statt CAS Consumer AEs verwenden
- CAS Consumer müssen die Schnittstelle `org.apache.uima.collection.CasConsumer` implementieren
- einfacher: Erweiterung der *Convenience base class* `org.apache.uima.collection.CasConsumer_ImplBase`

Entwicklung von *CAS Consumern*

- Note: Ab UIMA 2.x sind CAS Consumer spezielle AEs - In Zukunft sollte man statt CAS Consumer AEs verwenden
- CAS Consumer müssen die Schnittstelle `org.apache.uima.collection.CasConsumer` implementieren
- einfacher: Erweiterung der *Convenience base class* `org.apache.uima.collection.CasConsumer_ImplBase`
- Implementierung der abstrakten Methoden nötig
  - `initialize()`(nicht abstrakt): Resourcen initialisieren, Zugriff auf Parameter

Entwicklung von *CAS Consumern*

- Note: Ab UIMA 2.x sind CAS Consumer spezielle AEs - In Zukunft sollte man statt CAS Consumer AEs verwenden
- CAS Consumer müssen die Schnittstelle `org.apache.uima.collection.CasConsumer` implementieren
- einfacher: Erweiterung der *Convenience base class* `org.apache.uima.collection.CasConsumer_ImplBase`
- Implementierung der abstrakten Methoden nötig
  - `initialize()`(nicht abstrakt): Resourcen initialisieren, Zugriff auf Parameter
  - `processCas(CAS)`: Prozessiern des CAS, z.B. schreiben in eine XML-Datei

Entwicklung von *CAS Consumern*

- Note: Ab UIMA 2.x sind CAS Consumer spezielle AEs - In Zukunft sollte man statt CAS Consumer AEs verwenden
- CAS Consumer müssen die Schnittstelle `org.apache.uima.collection.CasConsumer` implementieren
- einfacher: Erweiterung der *Convenience base class* `org.apache.uima.collection.CasConsumer_ImplBase`
- Implementierung der abstrakten Methoden nötig
  - `initialize()`(nicht abstrakt): Resourcen initialisieren, Zugriff auf Parameter
  - `processCas(CAS)`: Prozessiern des CAS, z.B. schreiben in eine XML-Datei
- `batchProcessComplete()` und `collectionProcessComplete()`: optionale Methoden die vom Framework nach Beendigung eines *Batch* oder der *Collection* aufgerufen werden

Zugriff auf *Annotation Results*

Listing 3: Zugriff auf Annotations vorgeschalteter AEs

```
1 FSIndex timeIndex = aJCas.getAnnotationIndex(TimeAnnot.type);
2 Iterator timeIter = timeIndex.iterator();
3 while (timeIter.hasNext()) {
4   TimeAnnot time = (TimeAnnot)timeIter.next();
5
6   //do something
7 }
```

Zugriff auf die Index FS

Listing 4: Zugriff auf die Resultate von vorgeschalteten AEs

```
1  public void processCas(CAS aCAS) throws ResourceProcessException {
2
3      JCas jcas;
4      try {
5          jcas = aCAS.getJCas();
6      } catch (CASException e){
7          throw new ResourceProcessException(e);
8      }
9      FSIterator iterator jcas.getJFSIndexRepository()
10         .getAllIndexFS(DocumentMetadata.type);
11     if (iterator.hasNext()) {
12         DocumentMetadata docMetadata = (DocumentMetadata) iterator.next();
13         String documentURL = docMetadata.getDocumentURL();
14         // do more
15     }
16
17 }
```

Outline

Resource Manager

- Zum Zugriff auf externe Resourcen, wie z.B. Datenbanken, Files
- Resourcen weden deklariert mittels XML
- mehr siehe 1.5.4. von [2]

Deployment Modes

- Drei verschiedene *deployment* Modes für *CAS Processors* möglich:
  - *Intergrated*: Laufen alle in der gleichen JVM
  - *Managed*: Laufen alle in separaten Prozessen auf dem gleichen Rechner
  - *Non-Managed*: Laufen in separaten Prozessen, eventuell auf verschiedenen Rechnern
- Bei den letzteren beiden Möglichkeiten müssen CASes zwischen verschiedenen Prozessen (und Rechnern) transportiert werden - Hierfür dient *Vinci*, ein Kommunikationsprotokoll

CAS Multiplier

Aufgabe eines *CAS Multiplier*

CAS Multiplier werden genutzt, um die Aufteilung der Daten in eine Serie von CASes zu ändern

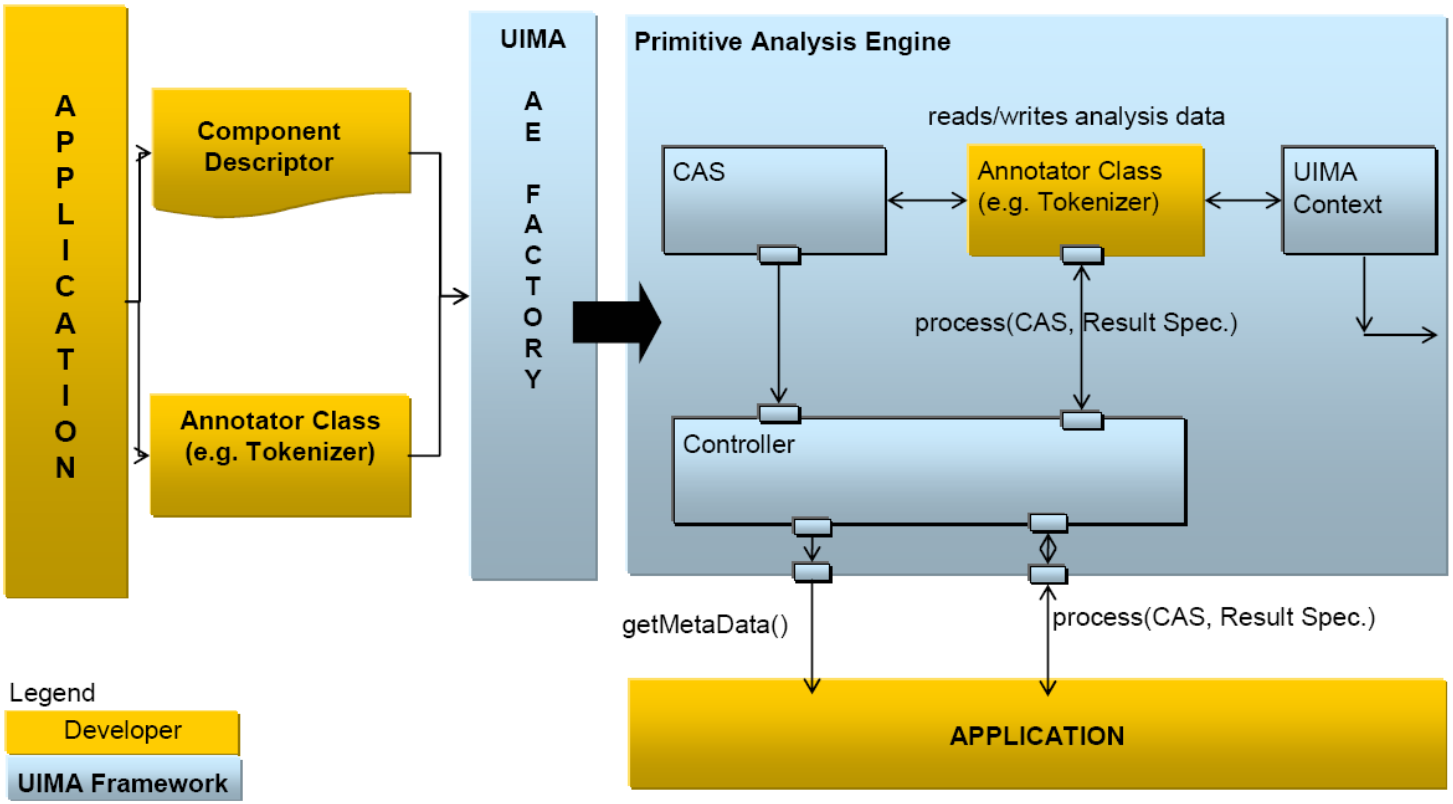
Beispiele:

- Splitten von großen Dokumenten in kleinere Einheiten, z.B. eine große pdf-Datei in einzelne Kapitel
- zum Vereinen von mehreren CASes zu einem CAS

Flow Controller

- Innerhalb von *Aggregate AEs* Fluss die Abarbeitung definiert werden
- Hierz dienen *Flow Controller*
- Einfachster Datenfluss: Hintereinabderschften von AEs in fester Reihenfolge
- aber auch komplexere Flüsse möglich

Anwendung von UIMA aus einer Anwendung (aus [1])



Outline

Lucas: Lucene CAS Indexer

- Brücke zwischen UIMA und Lucene: CAS-Consumer für Lucene
- Mapping für CASes zu Lucene `index documents`
- Wird konfiguriert über *description parameter* und ein *mapping file*


Apache-UIMA-Development-Community.
Uima overview & sdk setup.
2009.


Apache-UIMA-Development-Community.
Uima tutorial and developers guides - version 2.2.2.
2009.