NLP Processing with UIMA

Conceptual Guide for Parliamentary Speech Analysis

Group-8 Tuesday 14-16 PPR Tutor

Prepared by: Solomon Mengesha Kebede

This guide will help you understand the concepts you need to complete the assignment. It focuses on WHAT you need to do and WHY, not HOW to code it. You will develop the implementation yourself.

Table of Contents

1. Understanding UIMA and CAS

What is UIMA?

UIMA (Unstructured Information Management Architecture) is a framework designed to process unstructured content like text, audio, and video. It provides a standardized way to analyze documents and share annotations between different processing components.

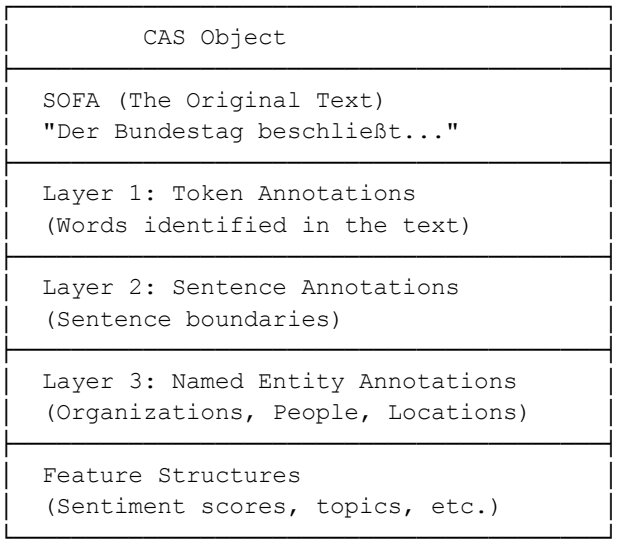
What is CAS?

CAS (Common Analysis Structure) is the central data structure in UIMA. Understanding CAS is crucial for this assignment.

The CAS contains:

- SOFA (Subject of Analysis) - The original text you want to analyze
- Annotations - Information extracted from the text (tokens, entities, etc.)
- Feature Structures - Additional properties and metadata
- Views - Different representations of the same document

Visualizing CAS Structure



💡 Key Concept

Think of CAS as a layered cake. The base layer is your text (SOFA). Each processing step adds a new layer of annotations on top. All layers reference the same original text.

🤔 *Think: Why do you think UIMA uses this layered approach instead of just storing results in separate variables?*

2. Converting MongoDB Data to CAS

Your first challenge is converting speech documents from MongoDB into CAS objects. This requires understanding how to map your data structure.

Step 1: Analyze Your Data

Look at your MongoDB documents. What information do they contain?

- Speech text (the main content)
- Speaker information (name, ID)
- Metadata (date, protocol ID)
- Video reference (URL or file path)

Step 2: Design the Mapping

Consider these mapping questions:

- Which field becomes the SOFA (the text to analyze)?
- How do you preserve speaker information in the CAS?
- Where should you store the MongoDB document ID for later reference?
- How do you handle optional fields like video URLs?

💡 Implementation Hint

Look up JCasFactory for creating CAS objects, and DocumentMetaData for storing document-level information. You may need to create custom annotation types for your specific metadata.

🤔 *Think: What happens if a speech has no text? How should your toCAS() method handle this?*

Step 3: The toCAS() Method

Your toCAS() method should:

- Create a new CAS object
- Set the document text from your MongoDB field
- Add metadata annotations
- Return the initialized CAS ready for processing

3. The NLP Pipeline Architecture

An NLP pipeline processes text through multiple stages. Each stage adds specific annotations to your CAS object.

Pipeline Flow

```
Raw Text Input
      ↓
Tokenization
      ↓
Sentence Detection
      ↓
POS Tagging
      ↓
Dependency Parsing
      ↓
Named Entity Recognition
      ↓
Sentiment Analysis
      ↓
Topic Classification
      ↓
Fully Annotated CAS
```

Understanding Each Component

Tokenization: Splits text into individual words and punctuation marks. Each token has begin and end positions.

> Example
> Text: "Der Bundestag beschließt."
> Tokens: "Der" (position 0-3), "Bundestag" (4-13), "beschließt" (14-24), "." (24-25)

Sentence Detection: Identifies sentence boundaries in the text.

POS (Part of Speech) Tagging: Labels each word with its grammatical category (noun, verb, adjective, etc.).

> Common German POS Tags
> NOUN - Nouns (Bundestag, Gesetz)
> VERB - Verbs (beschließt, diskutiert)

ADJ - Adjectives (wichtig, neu)
DET - Determiners (der, die, das)

Dependency Parsing: Shows grammatical relationships between words (which word depends on which).

🤔 *Think: Why might dependency information be useful for analyzing political speeches? Think about identifying who is doing what to whom.*

Named Entity Recognition (NER): Identifies and classifies real-world entities: people, organizations, locations, dates.

💡 Entity Types in German Politics
PER - Persons (Angela Merkel, Olaf Scholz)
ORG - Organizations (Bundestag, CDU, SPD)
LOC - Locations (Berlin, Deutschland)
DATE - Dates and times

Sentiment Analysis (GerVader): Measures the emotional tone of text: positive, negative, or neutral.

💡 Sentiment Scores
Scores range from -1 (very negative) to +1 (very positive)
Score of 0 indicates neutral sentiment

🤔 *Think: How might sentiment scores help you visualize political discourse patterns over time?*

Topic Classification (parlbert-topic): Categorizes the speech into policy areas (health, economy, education, etc.).

Integrating with Docker UIMA

The Docker Unified UIMA Interface provides these NLP tools as services. Your task is to:

- Send your CAS objects to the appropriate Docker services
- Receive back the annotated CAS with new layers added
- Ensure proper error handling for network issues
- Process documents efficiently (consider batching)

4. Designing with Object-Oriented Principles

Good design makes your code easier to test, maintain, and extend. The assignment requires "encapsulation in the sense of object orientation."

What Does Encapsulation Mean Here?

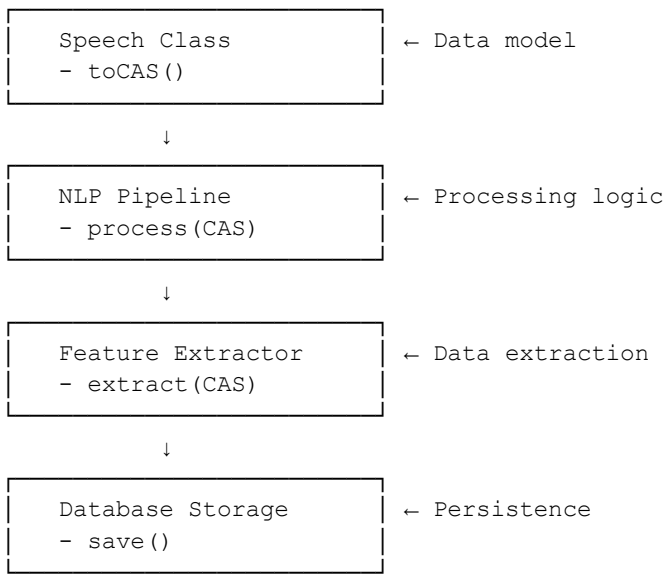
Keep related functionality together and hide implementation details:

- NLP processing logic should be separate from database operations
- Each processing component should be independent
- Use interfaces to define contracts between components
- Make it easy to swap implementations (e.g., different sentiment analyzers)

🤔 *Think: If you put all NLP processing in one giant method, what problems might you face when testing or debugging?*

Design Pattern: Separation of Concerns

Consider this structure:



💡 Design Tip

Create an interface (e.g., NLPProcessor) that defines a process() method. Each NLP component (spaCy, GerVader, parlbert) can implement this interface. This makes your pipeline flexible and testable.

Think About:

- How will you organize your classes in the nlp package?
- What methods should be public vs private?
- How can you make testing easier with your design?
- What happens if you want to add a new NLP component later?

5. Serialization: Why and How

NLP processing is computationally expensive and time-consuming. Running the pipeline on thousands of speeches could take hours or days.

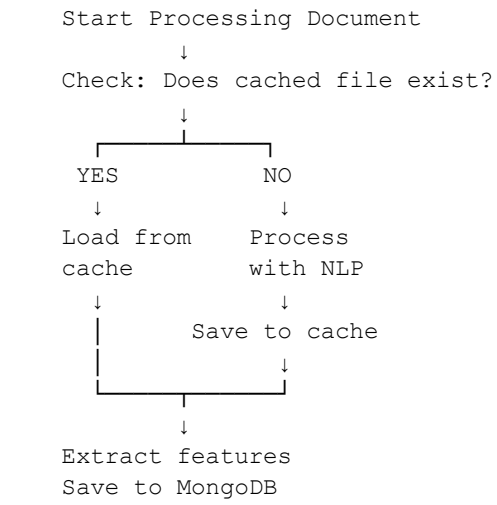
The Problem

Without serialization, you would re-process every document every time you run your program. This is wasteful and impractical.

The Solution: Serialize and Cache

After processing a document once, save the annotated CAS to disk. Next time, load it from cache instead of reprocessing.

Serialization Workflow



What You Need to Implement

- Check if a serialized CAS file exists for a document ID
- Serialize (save) an annotated CAS to disk in XMI format
- Deserialize (load) a CAS from disk back into memory
- Organize your cache directory (one file per document)
- Handle errors gracefully (corrupt files, missing files)

💡 UIMA Serialization Format
UIMA uses XMI (XML Metadata Interchange) format for serialization. Look for XmiCasSerializer and XmiCasDeserializer classes in UIMA documentation.

🤔 *Think: Where should you store cached CAS files? What file naming convention makes sense? How do you ensure you don't process the same document twice?*

Performance Consideration

With proper caching, your first run might take hours, but subsequent runs should take only seconds. This is the difference between practical and impractical.

6. Working with Multiple Views

UIMA Views allow you to store multiple representations of the same document in one CAS. This is crucial when you have both written speeches and video transcriptions.

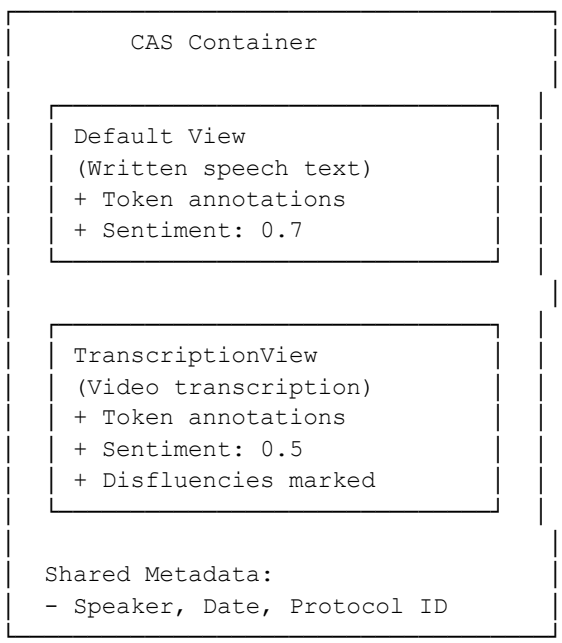
Why Use Views?

Consider these scenarios:

- Written speech: "Der Bundestag beschließt ein neues Gesetz."
- Spoken (transcribed): "Uh, der Bundestag, äh, beschließt also ein neues Gesetz."

These are two versions of the same content. You want to analyze both and compare them. Views let you keep both in one CAS object.

Multi-View Structure



View Operations

You need to learn how to:

- Create a new view with a specific name
- Set different text content for each view
- Process each view through the NLP pipeline

- Extract annotations from a specific view
- Compare results across views

💡 View Names

The default view is typically called "_InitialView". You can create additional views with any name, such as "TranscriptionView". Look up createView() in UIMA documentation.

🤔 *Think: How might you use views to compare formal written language vs. spontaneous spoken language? What differences would you expect to find in sentiment, vocabulary, or sentence structure?*

7. Video to Text Processing

The final piece is converting video recordings of speeches into text using Whisper, then analyzing that text with the same NLP pipeline.

Video Processing Pipeline

```
Video File
    ↓
Extract Audio
    ↓
Whisper Transcription
    ↓
Transcribed Text
    ↓
Add to CAS as New View
    ↓
Process with NLP Pipeline
    ↓
Compare with Written Version
```

Understanding Whisper

Whisper is an automatic speech recognition (ASR) model developed by OpenAI. It converts audio into text. For German parliamentary speeches, you'll want to:

- Specify German language for better accuracy
- Consider which model size to use (base = faster, large = more accurate)
- Handle long videos (Whisper works best on segments)
- Expect some transcription errors (background noise, overlapping speech)

Integration Steps

1. Download or access the video file
2. Extract audio track (you may need ffmpeg)
3. Call Whisper to transcribe the audio
4. Get the transcribed text
5. Create a new view in your existing CAS
6. Set the transcription as text for that view
7. Run the NLP pipeline on the transcription view

8. Serialize the CAS with both views

> 💡 Technical Note
> You can integrate Whisper via Python. Consider creating a Python script that your Java code can call, or use Whisper's API if available through Docker.

> 🤔 *Think: What challenges might arise when transcribing parliamentary speeches? Think about multiple speakers, formal language, and audio quality.*

Analyzing Transcriptions

Once transcribed, treat the text like any other text. Apply the same NLP pipeline (spaCy, GerVader, parlbert). The difference is that spoken language often contains:

- Disfluencies ("uh", "äh", "also")
- False starts and corrections
- More informal language
- Less perfect grammar

8. Your Implementation Roadmap

Here is a suggested order for completing the assignment. Each task builds on the previous ones.

Task (a): Create Package Structure

Create a new package called "nlp" in your project.

Questions to consider: How will you organize your classes? What belongs in the nlp package vs. other packages?

Tasks (b-c): CAS Conversion

Implement the toCAS() method in your Speech class.

- Fetch speeches from MongoDB
- Create CAS objects from speech data
- Map MongoDB fields to CAS structure (SOFA, metadata)
- Test with a few sample speeches

> 💡 Testing Tip
> Start with 5-10 speeches. Print out the CAS to verify it contains the expected data.

Task (d): NLP Pipeline Implementation

This is the core of the assignment. Set up your NLP pipeline.

- Configure Docker Unified UIMA Interface
- Create classes for each processor (SpaCy, GerVader, parlbert)
- Implement processing logic
- Test each component individually
- Chain components into a complete pipeline
- Ensure good object-oriented design (encapsulation!)

> 🤔 *Think: What should happen if one NLP component fails? Should you skip that document or continue with partial results?*

Task (e): Extract and Store Results

After processing, extract the results and save them to MongoDB.

- Extract tokens, sentences, POS tags from CAS
- Extract named entities
- Extract sentiment scores
- Extract topics
- Update the MongoDB document with these results

> 💡 Storage Design
> Think about how to structure the MongoDB document. Should you store tokens as an array? Should named entities be grouped by type? What format is most useful for visualization later?

Task (f): Serialization System

Implement caching to avoid reprocessing.

- Create a serialization utility class
- Implement save to XMI format
- Implement load from XMI format
- Check if document is already processed before starting
- Organize cache directory structure

Task (g): Interface Design

Create interfaces to generalize your design.

- Define an interface for NLP processors
- Ensure all processors implement this interface
- Use the interface in your pipeline manager
- Make it easy to add new processors later

> 🤔 *Think: What methods should your interface define? What do all processors have in common?*

Task (h): Feature Extraction Classes

Expand your class structure to cleanly extract features.

- Create dedicated methods for each feature type
- Ensure methods work with CAS annotations
- Consider creating a FeatureExtractor class
- Make extraction reusable and testable

Task (i): Video Transcription

Process videos and integrate transcriptions.

- Set up Whisper (local or via Docker)
- Download/access video files
- Extract audio from videos
- Transcribe audio to text
- Create TranscriptionView in CAS
- Process transcription with NLP pipeline
- Use multiple views correctly
- Compare written vs. spoken versions

Best Practices and Tips

✓ Development Strategy

- Start small - test with 5 speeches before processing thousands
- Test each component independently before integration
- Add logging to track progress and debug issues
- Handle errors gracefully - some speeches or videos may fail
- Use version control to track your changes
- Document your design decisions
- Write tests for critical components

✓ Common Pitfalls to Avoid

- Processing the same document multiple times (use serialization!)
- Mixing data access and NLP logic in one class (bad OO design)
- Forgetting to add annotations to indexes in CAS
- Not checking if videos exist before trying to process them
- Hardcoding file paths or configuration values
- Not handling character encoding properly (use UTF-8)
- Trying to process all documents at once (memory issues)
- Not validating CAS structure after deserialization

✓ Performance Optimization

- Use serialization to cache processed documents
- Process documents in batches to manage memory
- Consider parallel processing for independent documents
- Don't re-process if annotations already exist
- Monitor processing time and optimize bottlenecks

Learning Resources

Learning resources are available on your Exercise sheet and Olat Page.

Final Notes

This assignment combines many concepts: natural language processing, object-oriented design, data persistence, and multimedia processing. It's challenging but valuable.

> 💡 Learning Goals
>
> By completing this assignment, you will understand:
> • How NLP pipelines work in practice
> • The importance of proper software architecture
> • How to handle large-scale data processing
> • Integration of multiple tools and frameworks
> • The difference between written and spoken language

Good luck with your implementation!