Aid to spatial navigation within a UIMA annotation index

Nicolas Hernandez

LINA CNRS UMR 6241 – University de Nantes

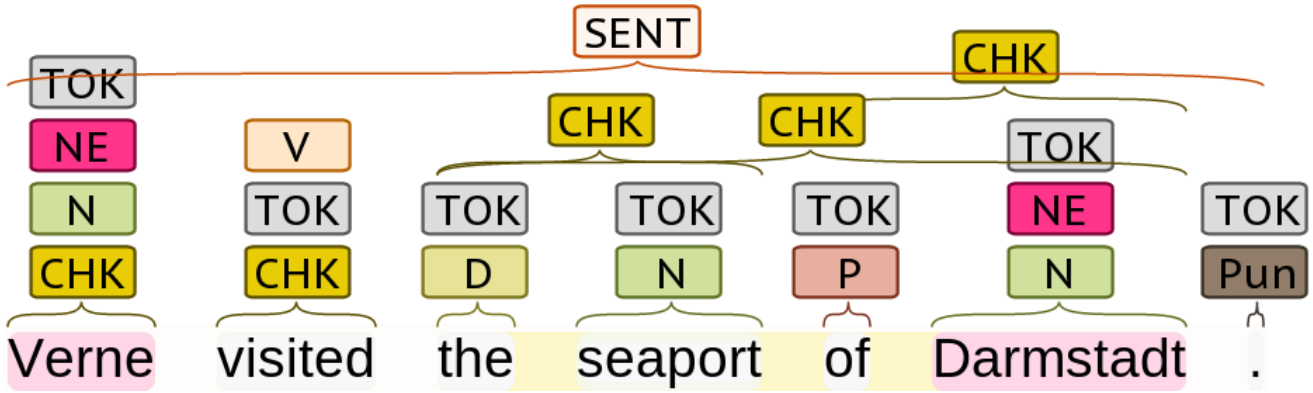
Darmstadt, 3rd UIMA@GSCL Workshop, September 23, 2013

UIMA is nice, but it is not easy to get into it

Motivation

- Supporting the Natural Language Processing developers
- when developing new analytics
- to access the analysis results produced by distinct components

Examples of access

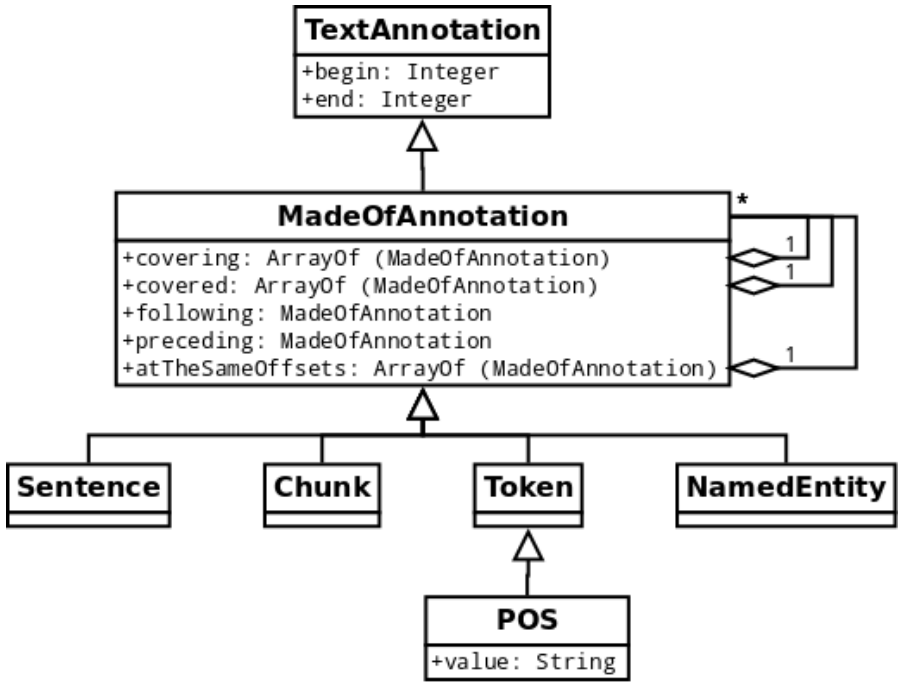


- Select the annotations verifying certain criteria (e.g. only `Nouns` and `Verbs`)
- Get the property of a textual object (e.g. is the current `Token` a `Verb` ?)
- Get the objects which are *part of* or *made of* other ones (e.g. the `Tokens` of a given `Sentence` or the `Sentence` of a given `Token`)
- Get the objects verifying syntactic constraints (e.g the `named entities` followed by a `verb` or all the objects contiguously located in the right context of another)

A simple solution

Define a domain model which make explicit all the structural links

In practice, in the context of UIMA, annotations are often produced by independently developed components without explicit relations between them (interoperability issue)



A domain model is called a type system in the UIMA jargon

The problem of accessing the annotations

Problem

We address the problem of accessing one annotation from another

- when the type system does not specify an explicit link between them
- and when annotations are related by a spatial relation

By spatial relation, we mean covering/covered, contiguously preceding/following and at the same offsets relations.

The problem of accessing the annotations

The problem depends on the means offered by the framework to index the annotations and to navigate within the index

`We argue that the Apache UIMA framework`

- is not enough intuitive
- has some restrictions which prevent from a complete spatial navigation among the annotations
- is forcing the way of developing algorithms

Related works

[Boguraev and Neff, 2010]

- Exposed a language for specifying spatial constraints among annotations
- Implemented a pattern matching engine with the UIMA which is no freely available
- May depend on specific configurations (type system and type priority)

Apache UIMA Ruta

- A freely available solution to perform pattern matching over annotations
- Integrates some navigation devices and uses the uimaFIT library
- Main difference with our approach : Ruta is an Analysis Engine by itself while our work aims at aiding to develop his own AE

Apache uimaFIT library

- Simplify the UIMA developments
- Holds methods to get the first annotation of a given type in whatever directions of the annotation space

Contributions

Our proposition relies on these works. In particular, it extends some uimaFIT abilities

In the following, I will

- Discuss the limitations of the Apache UIMA framework for supporting a spatial navigation within annotations
- Propose a more NLP-aware UIMA API (and discuss an implementation)

Means offered by the Apache UIMA framework

Definition: the built-in type, called `uima.tcas.Annotation`

begin and end features to attach the description to a specific region of the text

[Gurevych et al., 2007, Hahn et al., 2007, Kano et al., 2009]

Definition: the built-in annotation index, called `AnnotationIndex`

indexes the `uima.tcas.Annotation` (and subtypes of)

1. Sorted by increasing begin position (the longest ones come first)
2. and when occurring at the same offsets, possibility to define a *type priority*
   Or left undefined if unset

Means offered by the Apache UIMA framework

The API provides methods to get all the annotations of a given type (and subtypes)

Four methods are available to parse the index : Iterator, Subiterator, Tree and Constrained iterator

We will discuss the restrictions of these methods

Corresponding annotation index to our text example

Offsets	Annotations	Covered text
(0,37)	Document	*Verne visited the seaport of Darmstadt.\n*
(0,36)	Sentence1	*Verne visited the seaport of Darmstadt.*
(0,5)	Token1	*Verne*
(0,5)	NamedEntity1	*Verne*
(0,5)	N1	*Verne*
(0,5)	Chunk1	*Verne*
(0,0)	Source	
(6,13)	Token2	*visited*
(6,13)	V1	*visited*
(6,13)	Chunk2	*visited*
(14,35)	Chunk3	*the seaport of Darmstadt*
(14,25)	Chunk4	*the seaport*
(14,17)	Token3	*the*
(14,17)	D1	*the*
(18,25)	Token4	*seaport*
(18,25)	N2	*seaport*
(26,35)	Chunk5	*of Darmstadt*
(26,28)	Token5	*of*
(26,28)	P1	*of*
(29,35)	Token6	*Darmstadt*
(29,35)	NamedEntity2	*Darmstadt*
(29,35)	N3	*Darmstadt*
(35,36)	Token7	.
(35,36)	Pun1	.

- Annotations are identified by their type and an index number
- Order of addition in the index : first the `Document`, then the `Source`, the `Sentence`, the `Tokens`, the `POS`, the `Chunks` and the `NamedEntities`
- No type priority definition

Moving to the previous/next annotation in the index

(0,37)	Document
(0,36)	Sentence1
(0,5)	Token1
(0,5)	NamedEntity1
(0,5)	N1
(0,5)	Chunk1
(0,0)	Source
(6,13)	Token2
(6,13)	V1
(6,13)	Chunk2
(14,35)	Chunk3
(14,25)	Chunk4
(14,17)	Token3
(14,17)	D1
(18,25)	Token4
(18,25)	N2
(26,35)	Chunk5
(26,28)	Token5
(26,28)	P1
(29,35)	Token6
...	...

Method

Ambiguous iterator

Various relations between two contiguous annotations in the index:

- Covering/covered relation : e.g. Sentence1 then Token1
- At the same offsets : e.g. Token2 then V1
- Spatially contiguous (preceding/following) : e.g. Chunk2 then Chunk3
- No spatial relation : e.g. Chunk1 then Source

Use case : Iterate only over some selected types

Example

Only `Nouns` and `Verbs`

Solutions

- Create a super type and get an index of it → make a less consistent type system from a linguistic POV
- Use ambiguous iterator to parse the index and test the type of each annotation
- Build a constrained iterator by specifying the types to return when parsing the whole index → constraints are complex objects not always easy to handle

No limitation here but point out that a simple need may require complex mechanisms to be set

Use case : Iterate over spatially contiguous annotations

(0,37)	Document
(0,36)	Sentence1
(0,5)	Token1
(0,5)	NamedEntity1
(0,5)	N1
(0,5)	Chunk1
(0,0)	Source
(6,13)	Token2
(6,13)	V1
(6,13)	Chunk2
(14,35)	Chunk3
(14,25)	Chunk4
(14,17)	Token3
(14,17)	D1
(18,25)	Token4
(18,25)	N2
(26,35)	Chunk5
(26,28)	Token5
(26,28)	P5
(29,35)	Token6
...	...

Method

Unambiguous iterator
(get successively the first annotation in the index whose begin value is higher than the end of the current one)

Unexpected result

Called on the full annotation index, it starts from the first annotation in the index
E.g. here no more than the Document annotation

Unexpected result

Called on a typed annotation index, with (partially) overlapping annotations the developer may not access all the annotations
E.g. when iterating over the Chunk type, returns Chunk1, Chunk2 and Chunk3. Chunk4 and Chunk5 are not reachable

Testing the uimaFIT "selectFollowing of a given type" method

It only returns the `Chunk` 1 to 3, and misses the 4th and 5th, when calling it successively from the first chunk

Still some iterator limitations

Use case

Iterate in a reverse order over spatially contiguous annotations thanks to an unambiguous iterator

Unexpected result

If two overlapping annotations precede the current one, the one returned will be the one whose begin offset is the smallest and not the one with the highest end value (lower than the begin value of the current one)

Use case

No possibility to switch from an ambiguous iterator to an unambiguous one (and vice-versa) to iterate over the index and in the text spatiality in the same time

Use case : Get the covered annotations of a given one

(0,37)	Document
(0,36)	Sentence1
(0,5)	Token1
(0,5)	NamedEntity1
(0,5)	N1
(0,5)	Chunk1
(0,0)	Source
(6,13)	Token2
(6,13)	V1
(6,13)	Chunk2
(14,35)	Chunk3
(14,25)	Chunk4
(14,17)	Token3
(14,17)	D1
(18,25)	Token4
(18,25)	N2
(26,35)	Chunk5
(26,28)	Token5
(26,28)	P1
(29,35)	Token6
...	...

Method

Ambiguous subiterator

Unexpected result

Without a type priority definition, no assurance that the annotations at the same spans come in the expected conceptual order ; Or are even returned

Example

Over each chunk annotation for getting the Tokens returns

- the Source annotation for Chunk1,
- nothing for Chunk2,
- and the expected Tokens (and more to filter) for the all remaining Chunks (e.g. Chunk4)

Use case : Get the covered annotations of a given one

(0,37)	Document
(0,36)	Sentence1
(0,5)	Token1
(0,5)	NamedEntity1
(0,5)	N1
(0,5)	Chunk1
(0,0)	Source
(6,13)	Token2
(6,13)	V1
(6,13)	Chunk2
(14,35)	Chunk3
(14,25)	Chunk4
(14,17)	Token3
(14,17)	D1
(18,25)	Token4
(18,25)	N2
(26,35)	Chunk5
(26,28)	Token5
(26,28)	P1
(29,35)	Token6
...	...

Example

The Tokens of the Chunks

Methods

- Ambiguous subiterator + type priority (Chunks > Tokens) + test to select the desired types
- Ambiguous iterator and search backward/forward the index to find at the same offsets and covered + test to select types
- uimaFIT + a specific covered type to look for + test to select types (and filter subtypes)

Still some code to write

Get the covering annotation of a given annotation

`Tree method`

- Only give the descendants of a covering annotation → cannot be used to get the ancestors of a given annotation
  (a tree of the whole document is useless)
- No way to get a direct access to a node (the tree has to be parsed from the root)
- Built thanks to successive calls of unambiguous subiterators → partially overlapped annotations may not be present in the tree

uimaFIT may handle the specific case when searching a specific covering type

Here again an ambiguous iterator for searching the annotation index would be the solution

Missing methods

`No dedicated method for`

- *super-iterating* and to get the annotations covering a given annotation (partially handled by uimaFIT)
- moving to the first/last annotation of a given type or the preceding/following annotation of a given type (handled by uimaFIT)
- getting partially-covered (preceding or following) annotations

At this point we can say that

The UIMA framework

- Offers partial solutions for navigating in the annotation space
- May involve the combination of several specific mechanisms

All these remarks often lead the developers to

- use preferentially ambiguous iterators and subiterators,
- even if, this causes to write more code to search the index backward/forward and tests (on type and offsets) to select the desired annotations

uimaFIT may simplify a bit these developments

To support a spatial navigation among annotations

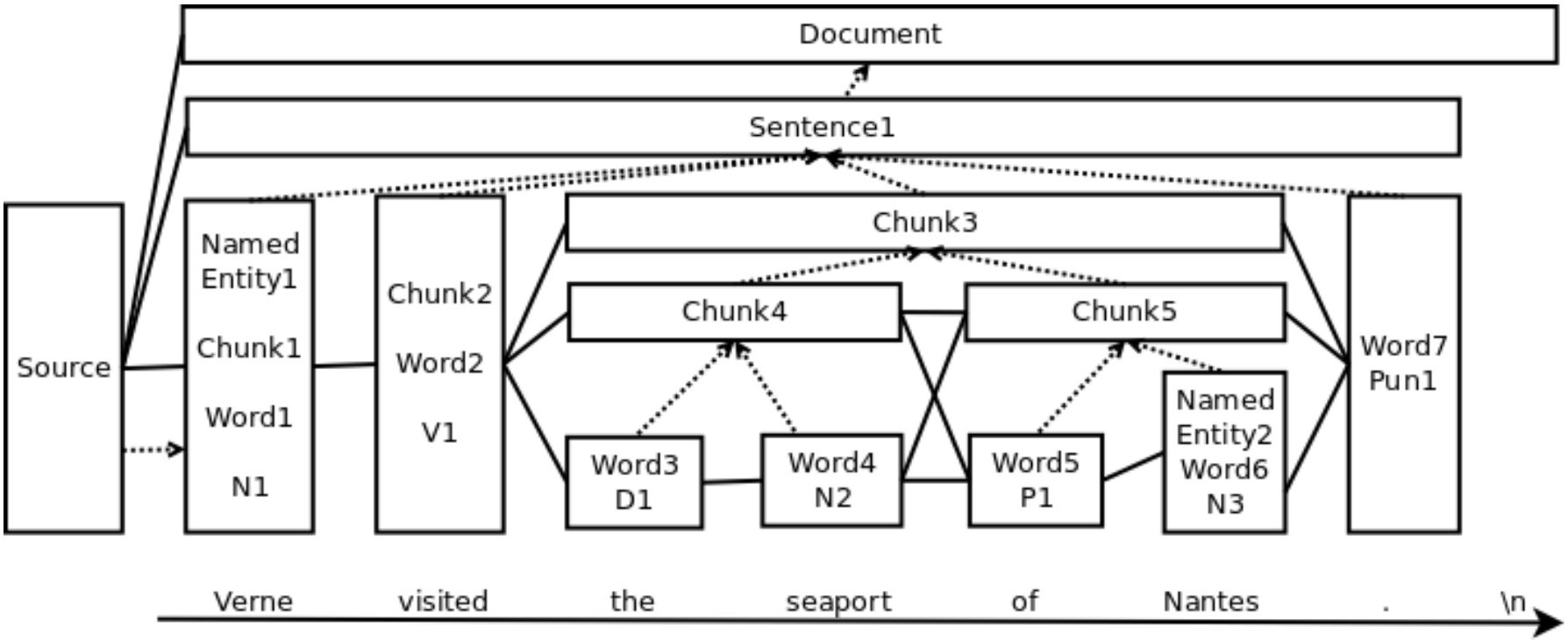
Proposition

1. Give access at a time to all the annotations present at a given span
2. Offer methods to explore the spatial vicinity of a given location

Definition: LocatedAnnotation

We define a specific structure to refer to all the annotations of a given location

Spatial relations which interconnect the LocatedAnnotations



- Boxes represent the `LocatedAnnotation` aligned on their text spans
- Solid lines represent the spatial preceding/following relation
- Dotted lines represent the parent/child relations (parent is indicated by an arrow)

An API for supporting spatial navigation

- Get access to a `LocatedAnnotation` thanks to a pair of offsets
- List the annotations of a given `LocatedAnnotation`
- Get the covering/(list of) covered/preceding/following `LocatedAnnotation` in the annotation space
- Get the (list of) partially preceding/following `LocatedAnnotation`
- Get the first ancestor/descendent/self `LocatedAnnotation` containing a given annotation type
- Get the first preceding/following `LocatedAnnotation` containing a given annotation type
- Get the first/last `LocatedAnnotation` containing a given annotation type
- Test if a `LocatedAnnotation` has a property and get its value

Current implementation of the API

Requires to pre-build the `LocatedAnnotations` (i.e. determine the annotations occurring at the same spans and identify the local vicinity of each of them)

- Memory consumption is not important
- But the time for building the structure is a problem
  about 2 seconds for a 50-sentences text analysed with sentences, chunks and Tokens
  $\rightarrow$ even optimized, it will always exist
- Mechanism for updating the structure
  $\rightarrow$can be developed but similar mechanism already exists in the native UIMA index
- Exchange the structure between components
  $\rightarrow$can be accomplished but requires the definition of a dedicated type system

Conclusion

The solution we propose aims both at

- Solving a case of interoperability issue (how to handle annotations with no type relation but with spatial relations)
- Facilitating the access to the UIMA data structure for NLP developer

A version is available at the `https://uima-mapper.google.com`

Next steps

- Implement the API without pre-calculating the `LocatedAnnotation`
  - to avoid the building time (this means to search the annotation index on the fly and probably has also a time cost)
  - will benefit from the updating mechanism of the `AnnotationIndex`,
  - and avoid the definition of specific type system to exchange data
- Submit to Apache, possibly as uimaFIT extensions

Questions ?

Have you already visited the seaport of Darmstadt ?


Boguraev, B. and Neff, M. S. (2010).
A framework for traversing dense annotation lattices.
Language Resources and Evaluation, 44(3) :183–203.


Gurevych, I., Mühlhäuser, M., Müller, C., Steimle, J., Weimer, M., and Zesch, T. (2007).
Darmstadt knowledge processing repository based on uima.
In *First Workshop on UIMA at GSCL*, Tübingen, Germany.


Hahn, U., Buyko, E., Tomanek, K., Piao, S., McNaught, J., Tsuruoka, Y., and Ananiadou, S. (2007).
An annotation type system for a data-driven nlp pipeline.
In *The LAW at ACL 2007*, pages 33–40.


Kano, Y., McCrohon, L., Ananiadou, S., and Tsujii, J. (2009).
Integrated NLP evaluation system for pluggable evaluation metrics with extensive interoperable toolkit.
In *SETQA-NLP*, pages 22–30.