Second Version of Pilot Applications

Deliverable D4.6

Version 1.0

2012-07-31

METANET4U

www.metanet4u.eu

The central objective of the Metanet4u project is to contribute to the establishment of a pan-European digital platform that makes available language resources and services, encompassing both datasets and software tools, for speech and language processing, and supports a new generation of exchange facilities for them.
This central objective is articulated in terms of the following main goals:
Assessment: to collect, organize and disseminate information that permits an updated insight into the current status and the potential of language related activities, for each of the national and/or language communities represented in the project. This includes organizing and providing a description of: language usage and its economic dimensions; language technologies and resources, products and services; main actors in different areas, including research, industry, government and society in general; public policies and programs; prevailing standards and practices; current level of development, main drivers and roadblocks; etc.
Collection: to assemble and prepare language resources for distribution. This includes collecting languages resources; documenting these language resources; upgrading them to agreed standards and guidelines; linking and cross-lingual aligning them where appropriate.
Distribution: to distribute the assembled language resources through exchange facilities that can be used by language researchers, developers and professionals. This includes collaboration with other projects and, where useful, with other relevant multi-national forums or activities. It also includes helping to build and operate broad inter-connected repositories and exchange facilities.
Dissemination: to mobilize national and regional actors, public bodies and funding agencies by raising awareness with respect to the activities and results of the project, in particular, and of the whole area of language resources and technology, in general.

METANET4U is a project in the META-NET Network of Excellence, a cluster of projects aiming at fostering the mission of META. META is the Multilingual Europe Technology Alliance, dedicated to building the technological foundations of a multilingual European information society.

META

METANET4U is co-funded by the participating institutions and the ICT Policy Support Programme of the European Commission

and by the participating institutions:

Faculty of Sciences, University of Lisbon

Instituto Superior Técnico

University of Manchester

University *Alexandru Ioan Cuza*

Research Institute for Artificial Intelligence, Romanian Academy

University of Malta

Technical University of Catalonia

Universitat Pompeu Fabra

Revision History

Version	Date	Author	Organisation	Description
0.7	20th July 2012	Sophia Ananiadou et al.	UNIMAN	First Draft
0.8	25th July 2012	Sophia Ananiadou et al.	UNIMAN	Second Draft incorporating feedback from partners
0.9	26th July 2012	Sophia Ananiadou et al.	UNIMAN	Pre-final version checked by reviewers
1.0	30th July 2012	Sophia Ananiadou et al.	UNIMAN	Final version

METANET4U

Second version of pilot applications

Document METANET4U-2012-D4.6
EC CIP project #270893

Deliverable D4.6
Completion: Final
Status: Submitted
Dissemination level: Restricted to other programme participants

Responsible: Sophia Ananiadou (WPS coordinator)

Contributing Partners: UNIMAN, UPF, FCUL, IST, UAIC, RACAI, UOM, UPC

Authors: Sophia Ananiadou, Paul Thompson, John McNaught, Jorge Vivaldi, Núria Bel, João Balsa, Rita Henriques, João Silva, Thomas Pellegrini, Isabel Trancoso, Gonçalo Simões, Rui Lageira, Ionut Pistol, Radu Simionescu, Alex Moruz, Radu Ion, Dan Tufis, Andrew Attard, Jan Joachimsen, Mike Rosner, Antonio Bonafonte, Asunción Moreno

Reviewers: Mike Rosner, Andrew Attard

Contents

1 Introduction

META-SHARE is the infrastructure that will be used to make available the language resources (LRs) being released as part of METANET4U (and related projects). Developers of third-party LRs are also being encouraged to make their LRs accessible via META-SHARE, thus making it a useful facility for all types of LR developers and users.

As has already been described in detail in *D4.4 First version of pilot applications*, developers of NLP applications often want to combine various basic language processing tools (e.g. sentence splitters, tokenisers, part-of-speech taggers, etc.) into pipelines, to carry out more sophisticated tasks, such as parsing, named entity recognition, etc. A common barrier to chaining such tools together is that they are not interoperable. For example, different LRs are implemented in different programming languages, and they may have different input/output formats, or use different data types. This means that the developer may have to waste time writing extra code to ensure that different tools can "talk" to each other.

In order to address such issues, a major focus of work in the METANET4U project is to demonstrate explicitly the advantages that can be brought to NLP application building by making resources interoperable. Through our "pilot" work on interoperability, our aim is to encourage providers of LRs in META-SHARE to make their resources interoperable.

Interoperability can be achieved in a number of different ways. For example, the PANACEA project[1], with which one of the METANET4U project partners (UPF) is involved, ensures interoperability between LRs by making them available as web services which communicate via a common interface. PANACEA was described in more detail in *Deliverable D2.2 Specification of pilot services and applications.*

In METANET4U, our work on interoperability (Ananiadou et al, 2011; Thompson et al., 2011) makes use of another framework for interoperability that has already been adopted by a large number of language processing groups world-wide for making their LRs interoperable, i.e., the UIMA (Unstructured Information Management Architecture) framework (Ferrucci et al., 2006). The advantages of using UIMA include not only the fact that it is well-known, but also that a development environment, U-Compare (Kano et al, 2009; Kano et al, 2011) is available, which provides a graphical user interface allowing interoperable UIMA components to be combined together into workflows using simple drag-and-drop actions, and evaluated against gold standard corpora, with no coding effort required. Thus, by making LRs available as

[1] http://www.panacea-lr.eu/

UIMA components, and using them within U-Compare, it is possible to build and experiment with prototype applications, using various combinations of LRs, in a rapid and straightforward manner. This means that experiments with different configurations of workflows can be carried out in a straightforward manner by both experienced developers and less technical users. U-Compare and UIMA have both been described in detail in Deliverables 2.2 and 4.4.

The work on interoperability in the METANET4U project consists of the following:

a) Converting a subset of the LRs that METANET4U is making available on META-SHARE into interoperable UIMA components. These components will include both monolingual LRs operating on the various different languages involved in the project, as well as multi-lingual and speech-based LRs.
b) Using U-Compare, the newly created UIMA components are being combined in various ways (sometimes with existing components in the U-Compare library) to create a set of "pilot" applications or *workflows*, which can be used to carry out various different NLP tasks, often in a number of different languages. The purpose of this work is to showcase the power of UIMA and U-Compare in allowing various workflows to be constructed and evaluated easily and rapidly.

2 Interoperability work schedule

In *D4.4 First Version of Pilot Applications*, we provided a detailed description of the steps involved in our work on interoperability. Here, we provide a brief recap of these steps, together with information about current progress, and where to find information about the completed steps of the work.

1) **Identifying a set of LRs to make available as UIMA components.** These mainly consist of a subset of the LRs (both tools and corpora) which each partner agreed to upgrade and make available on META-SHARE, as specified in D*2.1 Report on first selection of resources*. The selection was determined according to the perceived potential to be incorporated within NLP and text mining workflows. To these were added some additional resources, in order to better demonstrate the potential for multi-lingual and speech-based applications within U-Compare. A total of 67 potentially suitable resources for use within UIMA workflows were identified, operating on 9 different languages, including 16 that had previously been made available as UIMA components by UNIMAN. The resources are listed and described in *Deliverable D2.2 Specification of Pilot Services and Applications.*

2) **Designing a set of workflows that make use of the UIMA-wrapped components.** UIMA components can be combined flexibly into workflows. In order to showcase this flexibility and versatility, a total of 26 specific workflows that make use of the wrapped components were designed, which are specified in several workflows *Deliverable D2.2 Specification of Pilot Services and Applications.* These workflows have the following purposes. Many of the workflows constitute multi-stage processes that are fundamental to many NLP applications (e.g. POS tagging, parsing, named entity recognition) and thus form building blocks that can be reused, extended and adapted in the creation of several different applications.

3) **Wrapping of the UIMA components.** This constitutes the first stage in the implementation of the workflows. The wrapping process consists of writing code to ensure that input/output of the selected LRs is handled in the way required by UIMA. At the heart of the UIMA framework is a common data structure called the *Common Analysis System (CAS)*, which can be accessed by all resources at the workflow. Each UIMA component must obtain its input by reading annotations (which may correspond to paragraphs, sentences, tokens, etc.) from the CAS. Output from components is stored by writing new annotations to the CAS, or else updating existing annotations. *Deliverable 4.4 First Version of Pilot Applications* provides more detail about the wrapping process, as well as providing implementation details of the 15 resources that were newly wrapped as UIMA components between M6 and M12, to complement the 16 resources already wrapped by UNIMAN that had also been identified as suitable for inclusion within workflows. In this report, we provide details of the 20 new resources that have been made available as UIMA components between M13 and M18. UNIMAN has continued to provide support to partners in creating their UIMA components.

   In order to ensure as much that the UIMA components created during the METANET4U project are as interoperable as possible, all input and output annotation types should be compatible with the U-Compare type system, which is a hierarchy of annotation types that are suitable to describe inputs/outputs of many NLP resources. Ensuring that all components use this sharable system of types ensures that the various components can "understand" each other's input and output annotations. U-Compare's existing library of around 50 components are already all compliant with this type system. More details about the U-Compare type system can be found in D2.2 and D4.4.

4) **Implementation of the workflows.** The workflows specified in *Deliverable D2.2 Specification of Pilot Services and Applications* act as a guide to how the various wrapped UIMA components can be combined together into workflows to carry out a number of useful NLP

tasks. In most cases, a particular task can be achieved in several different ways by using the available LRs in different combinations. As explained above, experimentation with different versions of workflows can be undertaken easily by building them using U-Compare's workflow canvas. However, we are also making available a number of sample "implemented" workflows, for each language and task. These take the form of special files containing ready-made workflows that can be imported into U-Compare and used immediately. Such workflows can act as "templates" for carrying out a particular task, that can be modified (e.g., by substituting alternative components) and extended as required. In *Deliverable 4.4 First Version of Pilot Applications*, we provided a detailed description of how workflows are created using the U-Compare workbench, and reported on the 10 workflows that could be implemented using the first batch of released UIMA components, at least for one or more of the planned languages. In the current deliverable, we report on the 8 additional workflows that can now be implemented, now that a second batch of UIMA components. Certain of the 10 original workflows can now be implemented in a greater number of languages than was possible at M12, when *Deliverable 4.4 First Version of Pilot Applications* was delivered. Since a number of annotated corpora are being delivered in this second batch of components, we also provide details about how evaluation workflows are created.

5) **Integration into the digital exchange platform.** It is planned to integrate most of the components into the "core" U-Compare library in the next stage of the project (where the licences of the individual components permit this). This will mean that, instead of the current method of downloading and importing the new components individually, most of the components will be available immediately by downloading U-Compare via the META-SHARE sight, making it extremely straightforward to create workflows in several different languages. Components with licences that are not compatible with the U-Compare licence will be available to download separately to import into U-Compare.

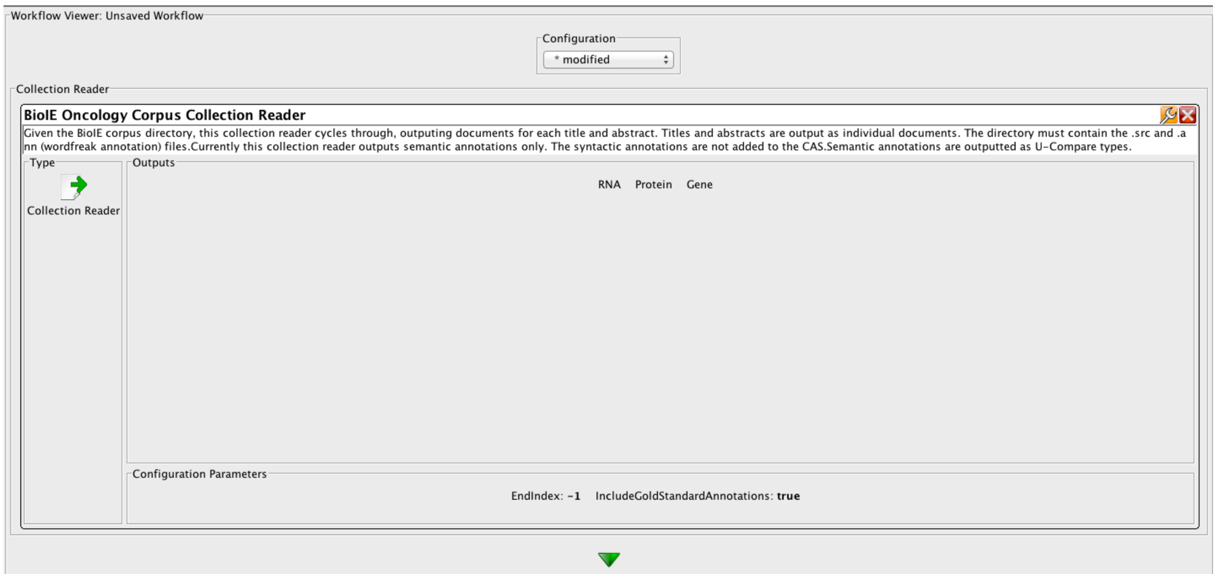
   The integration phase taking place between M19 and M24 will also involve making some enhancements to U-Compare itself. Certain components are reliant on these extensions, whilst other workflows will benefit from these extensions, such as the ability to see different "views" of a text side-by-side, e.g. source and target languages in a machine translation workflow, or a full text and a summary in a summarisation workflow. In addition, U-Compare can currently only handle the construction of workflows that process one document at a time and then move on to the processing of the next document. This model is not suitable for certain, more complex workflows, such as producing a single summary from multiple documents. The UIMA framework supports the merging of CASes of multiple documents, which would help to solve this problem. Extending U-Compare to permit the construction of such workflows is another area that will be explored during the final phase of the project.

3 Evaluation workflows in U-Compare

In Deliverable D4.4, detailed information was provided about how the 2 basic steps involved in the implementation phase of interoperability work are carried out, i.e. creating UIMA wrappers for components and using U-Compare to create U-Compare workflows. Information about creating specific types of workflows, i.e. evaluation workflows, was, however, left out of that deliverable. This was because evaluation workflows require that gold standard annotated corpora are available as UIMA components and, in Deliverable 4.4, no such components were made available. However, as part of this deliverable, 4 annotated corpora (3 English and 1 Portuguese) have been made available as UIMA components. These corpora allow 4 different evaluation workflows to be created, as detailed in section 5. In this section, we provide details regarding the creation of evaluation workflows. This section assumes some knowledge about how to create basic workflows in U-Compare; this information can be found in Deliverable D4.4.

Evaluation workflows compare (some of) the output annotations of one of more workflows against the annotations present in a gold standard annotated corpus. In order to do this, the annotations in the corpus must be represented as annotations in the CAS, in the same way as the output of the workflow(s), hence the need for UIMA corpus reader components, which convert annotations in the corpus into CAS format.

The first step in creating an evaluation workflow is to drag an appropriate corpus reader component onto the "Collection Readers" section of the U-Compare workflow canvas. Figure 1 shows an example of this. The "Outputs" section of the corpus information pane shows the types of annotations contained within the corpus.


Workflow Viewer: Unsaved Workflow

Configuration: * modified

Collection Reader

BioIE Oncology Corpus Collection Reader

Given the BioIE corpus directory, this collection reader cycles through, outputing documents for each title and abstract. Titles and abstracts are output as individual documents. The directory must contain the .src and .ann (wordfreak annotation) files.Currently this collection reader outputs semantic annotations only. The syntactic annotations are not added to the CAS.Semantic annotations are outputted as U-Compare types.

Type: Collection Reader

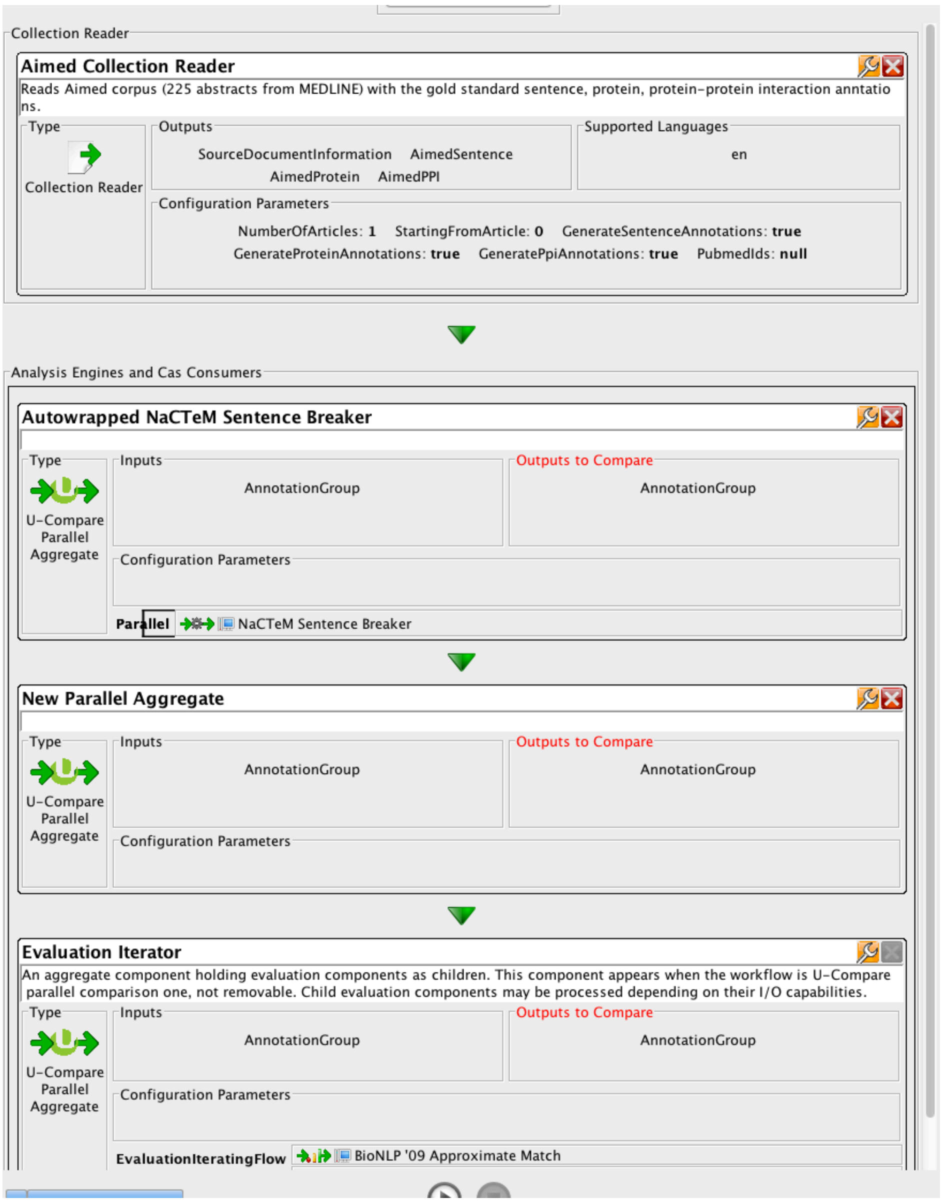
Outputs: RNA Protein Gene

Configuration Parameters: EndIndex: -1 IncludeGoldStandardAnnotations: **true**

Figure 1: A corpus collection reader component in U-Compare

Evaluation workflows allow the outputs of several different configurations of a workflow to be compared against the gold standard corpus, e.g. it is possible to compare the outputs of one or more different POS taggers against a corpus with gold standard POS annotations, using only a single workflow.

Any components that are required as pre-processing steps to create the types of annotations to be compared may be added to the workflow canvas as normal. However, the component(s) whose outputs are to be compared must be contained within a special type of component, a "Parallel Aggregate" component, which should be dragged onto the workflow canvas from the U-Compare library. Dragging this component onto the workflow canvas also causes an "Evaluation Iterator" component to be added automatically, as illustrated in Figure 2.



Collection Reader

Aimed Collection Reader

Reads Aimed corpus (225 abstracts from MEDLINE) with the gold standard sentence, protein, protein-protein interaction anntatio ns.

Type: Collection Reader

Outputs: SourceDocumentInformation AimedSentence AimedProtein AimedPPI

Supported Languages: en

Configuration Parameters: NumberOfArticles: **1** StartingFromArticle: **0** GenerateSentenceAnnotations: **true** GenerateProteinAnnotations: **true** GeneratePpiAnnotations: **true** PubmedIds: **null**

Analysis Engines and Cas Consumers

Autowrapped NaCTeM Sentence Breaker

Type: U-Compare Parallel Aggregate

Inputs: AnnotationGroup

Outputs to Compare: AnnotationGroup

Configuration Parameters

Parallel NaCTeM Sentence Breaker

New Parallel Aggregate

Type: U-Compare Parallel Aggregate

Inputs: AnnotationGroup

Outputs to Compare: AnnotationGroup

Configuration Parameters

Evaluation Iterator

An aggregate component holding evaluation components as children. This component appears when the workflow is U-Compare parallel comparison one, not removable. Child evaluation components may be processed depending on their I/O capabilities.

Type: U-Compare Parallel Aggregate

Inputs: AnnotationGroup

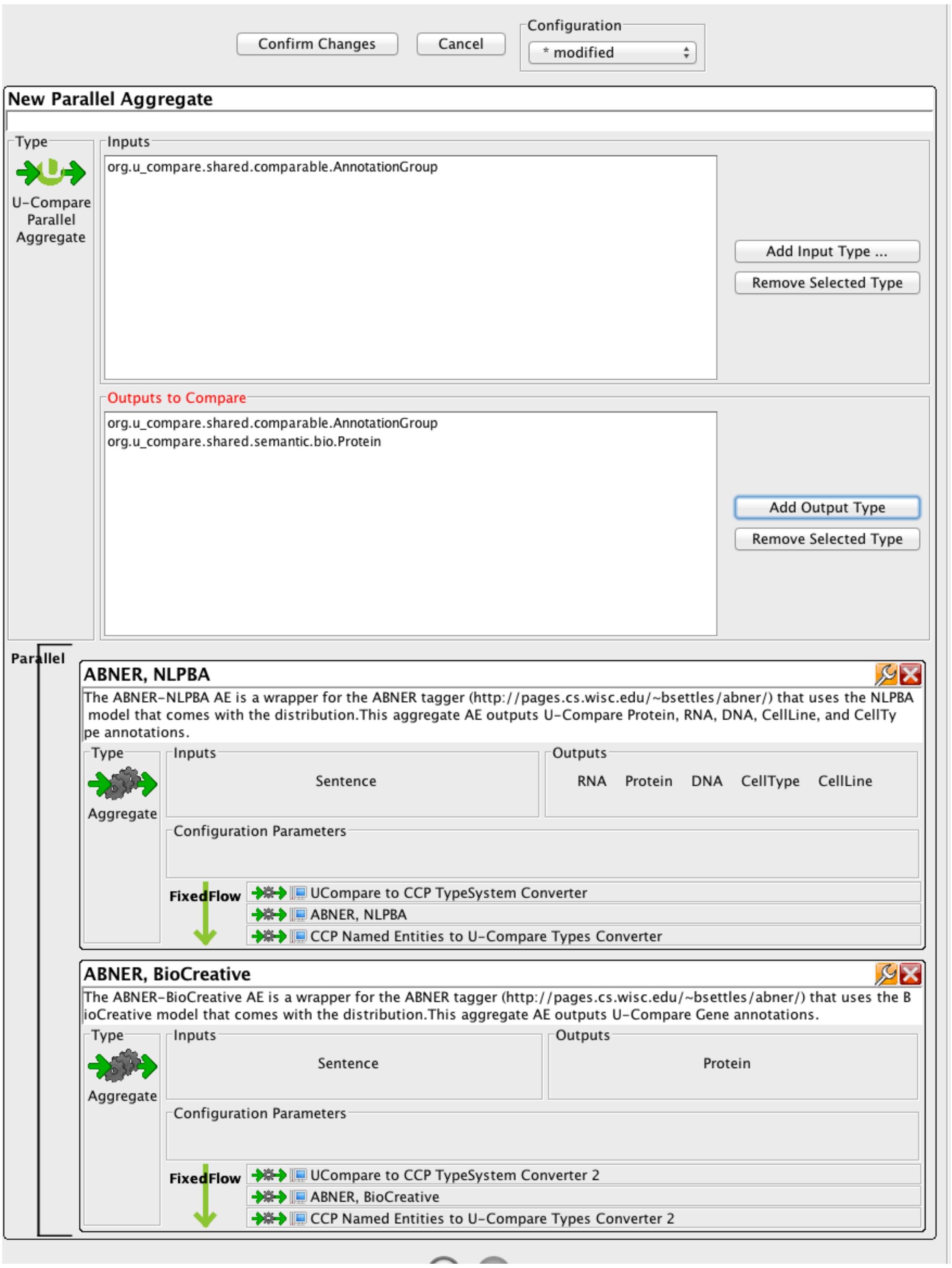
Outputs to Compare: AnnotationGroup

Configuration Parameters

EvaluationIteratingFlow BioNLP '09 Approximate Match

Figure 2: Evaluation Workflow

In order to perform comparison of annotations, the “Parallel Aggregate” component must be configured in two ways. Figure 3 shows the configuration window for this component, which is obtained by clicking on its icon.



New Parallel Aggregate

Type: U-Compare Parallel Aggregate

Inputs:
org.u_compare.shared.comparable.AnnotationGroup

Outputs to Compare:
org.u_compare.shared.comparable.AnnotationGroup
org.u_compare.shared.semantic.bio.Protein

ABNER, NLPBA

The ABNER-NLPBA AE is a wrapper for the ABNER tagger (http://pages.cs.wisc.edu/~bsettles/abner/) that uses the NLPBA model that comes with the distribution.This aggregate AE outputs U-Compare Protein, RNA, DNA, CellLine, and CellTy pe annotations.

Type: Aggregate
Inputs: Sentence
Outputs: RNA Protein DNA CellType CellLine
Configuration Parameters

FixedFlow:
- UCompare to CCP TypeSystem Converter
- ABNER, NLPBA
- CCP Named Entities to U-Compare Types Converter

ABNER, BioCreative

The ABNER-BioCreative AE is a wrapper for the ABNER tagger (http://pages.cs.wisc.edu/~bsettles/abner/) that uses the B ioCreative model that comes with the distribution.This aggregate AE outputs U-Compare Gene annotations.

Type: Aggregate
Inputs: Sentence
Outputs: Protein
Configuration Parameters

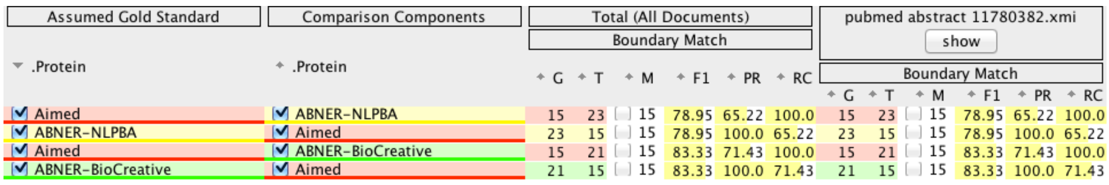
FixedFlow:
- UCompare to CCP TypeSystem Converter 2
- ABNER, BioCreative
- CCP Named Entities to U-Compare Types Converter 2

Figure 3: Configuration of parallel aggregate component

The section of the configuration window labelled “Parallel” allows U-Compare components to be dragged into it. The first step of the configuration is to drag one or more components into this section, whose output annotations will be compared against the gold standard annotations of the annotated corpus. In figure 3, two different named entity recognition tools have been dragged onto the component.

The second step of the configuration is to specify in the Parallel Aggregate component which type(s) of annotations should be compared. This is done by clicking on the "Add Output Type" button within the "Outputs to Compare" section. This causes a type system viewer window to be displayed, from which the appropriate annotation type(s) can be chosen by clicking on them. In figure 3, the type org.u_compare.shared.semantic.bio.Protein has been chosen for comparison, i.e., protein named entity annotations that are output by the tools will be compared with those in the annotated corpus.

Once the configuration of the Parallel Aggregate component is complete, the workflow can be run. In the results window, a special table is displayed that compares the outputs of the relevant tool(s) and the annotated corpus. An example of this table is shown in figure 4.

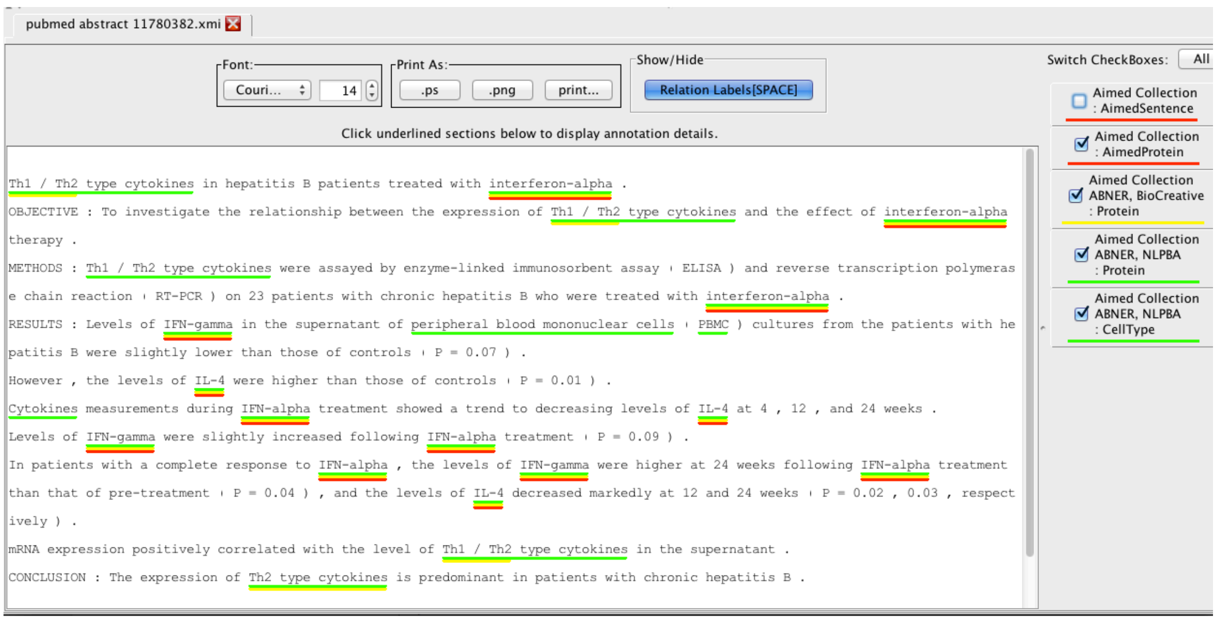


Assumed Gold Standard	Comparison Components	Total (All Documents)						pubmed abstract 11780382.xmi					
		Boundary Match						Boundary Match					
.Protein	.Protein	G	T	M	F1	PR	RC	G	T	M	F1	PR	RC
✓ Aimed	✓ ABNER-NLPBA	15	23	15	78.95	65.22	100.0	15	23	15	78.95	65.22	100.0
✓ ABNER-NLPBA	✓ Aimed	23	15	15	78.95	100.0	65.22	23	15	15	78.95	100.0	65.22
✓ Aimed	✓ ABNER-BioCreative	15	21	15	83.33	71.43	100.0	15	21	15	83.33	71.43	100.0
✓ ABNER-BioCreative	✓ Aimed	21	15	15	83.33	100.0	71.43	21	15	15	83.33	100.0	71.43

Figure 4: Output of comparison workflow

U-Compare produces pairwise comparisons of the corpus and tools, with different resources being assumed as the gold standard. Only certain rows in the table are relevant, i.e. those in which the annotated corpus appears in the "Assumed Gold Standard" column. In each row, several pieces of information are shown: the number of relevant annotations in the gold standard corpus (G), the number of annotations produced by the relevant tool (T), the number of matching annotations (M), the F1 score, precision (PR) and recall (RC). These figures can be viewed or the corpus as a whole, as well as for the individual documents within the corpus.

In order to compare the outputs of the tool(s) with the corpus annotations in more detail, the annotation viewer on the same screen allows the annotations produced by the different tools and in the annotated corpus to be viewed together. This is illustrated in Figure 5.


pubmed abstract 11780382.xmi
Font:
Couri...
14
Print As:
.ps
.png
print...
Show/Hide
Relation Labels[SPACE]
Switch CheckBoxes:
All
Click underlined sections below to display annotation details.
Aimed Collection : AimedSentence
Aimed Collection : AimedProtein
Aimed Collection ABNER, BioCreative : Protein
Aimed Collection ABNER, NLPBA : Protein
Aimed Collection ABNER, NLPBA : CellType

```
Th1 / Th2 type cytokines in hepatitis B patients treated with interferon-alpha .
OBJECTIVE : To investigate the relationship between the expression of Th1 / Th2 type cytokines and the effect of interferon-alpha
therapy .
METHODS : Th1 / Th2 type cytokines were assayed by enzyme-linked immunosorbent assay ( ELISA ) and reverse transcription polymeras
e chain reaction ( RT-PCR ) on 23 patients with chronic hepatitis B who were treated with interferon-alpha .
RESULTS : Levels of IFN-gamma in the supernatant of peripheral blood mononuclear cells ( PBMC ) cultures from the patients with he
patitis B were slightly lower than those of controls ( P = 0.07 ) .
However , the levels of IL-4 were higher than those of controls ( P = 0.01 ) .
Cytokines measurements during IFN-alpha treatment showed a trend to decreasing levels of IL-4 at 4 , 12 , and 24 weeks .
Levels of IFN-gamma were slightly increased following IFN-alpha treatment ( P = 0.09 ) .
In patients with a complete response to IFN-alpha , the levels of IFN-gamma were higher at 24 weeks following IFN-alpha treatment
than that of pre-treatment ( P = 0.04 ) , and the levels of IL-4 decreased markedly at 12 and 24 weeks ( P = 0.02 , 0.03 , respect
ively ) .
mRNA expression positively correlated with the level of Th1 / Th2 type cytokines in the supernatant .
CONCLUSION : The expression of Th2 type cytokines is predominant in patients with chronic hepatitis B .
```

Figure 5: Comparison of annotations in the annotation viewer

4 UIMA components delivered

In this section, we provide details of the resources that are included in the second release of wrapped UIMA components that accompanies this report. These resources have been wrapped between M13 and M18 of the project. We also include details about a few resources that were made available in the first batch of components, but whose implementation/functionality has been changed or enhanced between M18 and M24.

In general terms, the new language processing components do not cover extra languages, compared to those released in the first batch. Rather, they build upon the first batch of components, either by allowing more sophisticated language processing functionality for a given language, or else by providing alternative tools to carry out a specific type of processing that was provided by one of the tools in the first batch. Having alternative tools to carry out a particular language processing step is advantageous, since performance of workflows will often vary, according to which tools are combined together.

Two of the provided tools stand out as being slightly different from the others. The TTL-LangID does not add any annotations to the CAS, but rather can identify text as belonging to one of 54 different modern languages. This can be useful for more fully automated workflows, where the user does not have to specify the language to be used, where this would normally be expected as a configuration parameter. The E-txt2db tool is not a UIMA component itself, but rather a tool that *generates* UIMA components. E-txt2db is a language-independent trainable named entity recognition system. The tool provided performs both training of models and automatic generation of UIMA components using the models, making it extremely straightforward to integrate different trained models into workflows.

The provision in this batch of 4 UIMA components that are able to read annotated corpora into the CAS allows for the implementation of a number of evaluation workflows for English and Portuguese. Each of the three corpora (GENIA, GENIA event and GREC corpora for English, CINTIL corpus for Portuguese) contain different types of syntactic and/or semantic annotations. These act as gold standard annotations, against which annotations produced by various workflows can be compared and evaluated.

The UIMA-wrapped resources provided as part of this deliverable are in the form of Java Archive (jar) files or web service descriptors, which can easily be imported and tested in U-Compare. As mentioned above, many of these components will be integrated into U-Compare's core library during the final phase of the project, to make them even easier to use. For now, importing must be carried out, following the steps outlined in D4.4.

In the remainder of this section, we provide details of the 20 resources that constitute this second delivery of UIMA components. The resources are grouped according to the contributing partner, and in the same way as in D4.4, the following details are provided for each resource

- Brief description of the resource
- Languages handled/covered by the resource
- Input/output data types of the resource
- Corresponding U-Compare types used for input/output in the wrapped component. In the case that one or more of the types is an extension (subtype) of a type in the core U-Compare type system, this is indicated.
- Details about the U-Compare types used. These details include, e.g., whether the original type system was sufficient without modifications, whether the types used in the implemented component are different from those envisaged in D2.2, which new subtypes were created, and why, etc.
- Any relevant details or issues regarding the implementation of the wrapper code.

Three components provided in the first release of UIMA components, but whose implementation details have changed since the first batch, are also listed, together with the changes in implementation that have been carried out.

The jar files and web service descriptors corresponding to the newly wrapped resources have been uploaded onto the METANET4U intranet at the following location:

http://metanet4u.eu/intranet/index.php/WPS_-_Webservices_%28coord:_Sophia%29#Currently_wrapped_components

Components that were made available in the first release, as detailed in D4.4, are also available at the same location.

4.1 University of Lisbon (ULX)

4.1.1 Tools

LX-Tokenizer (Updated from D4.4)

Description: Splits sentences into tokens. In addition, it expands contractions, marks spacing around punctuation or symbols, detaches clitic pronouns verbs, and handles ambiguous strings.
Languages covered: Portuguese
Original resource implementation: Web service
Input: Sentences
U-Compare input type: `org.u_compare.shared.syntactic.Sentence`
Output: Tokens, with additional information as described above
U-Compare output type:
`pt.ul.fc.di.nlx.LXToken` (subtype of `org.u_compare.shared.syntactic.POSToken`)
U-Compare type details: The original UIMA wrapped component, made available as part of D4.4, used the output annotation type `org.u_compare.shared.syntactic.Token`. This annotation type is only sufficient to encode basic token information. However, the LX-Tokenizer web service outputs richer information than only basic tokens. Some examples of this extended functionality include expanding contractions and detaching clitic pronouns from verbs. In order for this information not to be lost in the U-Compare component, the component has been updated during this second phase of implementation. A new annotation type has been created, i.e., `pt.ul.fc.di.nlx.LXToken`, which is an extension of the annotation type `org.u_compare.shared.syntactic.POSToken` (since the extra information stored is also used by the LX-Tagger component, which adds POS tags to the LXToken annotations. The `pt.ul.fc.di.nlx.LXToken` type adds a new string-valued attribute, `expadnedForm`, which allows the full forms of words that have undergone some sort of contraction to be stored as part of the token annotation.

This information is required by the LX-Tagger component, and so its storage is needed in order to create more complex workflows for Portuguese.
Implementation details/issues: The tool is implemented as a web service, LXService. In order to implement the U-Compare component, the web service is invoked through a package, available as lxServiceClient.jar, which must be placed in the classpath of U-Compare. Its constructor requires one parameter related to the authentication of the client, namely the client's username, as this is registered at the LXService database of clients. We have created a new user in this database for U-Compare. The output of the call to web service is then converted to the format required by U-Compare.

<u>LX-POSTagger</u>

Description: Assigns part-of-speech tags to tokens. Requires information about the full forms of contracted words, as produced by the LX-Tokenizer component. Hence, the LX-Tokenizer must be run prior to this component in a workflow.
Languages covered: Portuguese
Original resource implementation: Web service
Input: Tokens in which contracted forms have been expanded
U-Compare input type: `pt.ul.fc.di.nlx.LXToken` (subtype of `org.u_compare.shared.syntactic.POSToken`)
Output: Tokens, with parts-of-speech assigned
U-Compare output type:
`pt.ul.fc.di.nlx.LXToken` (subtype of `org.u_compare.shared.syntactic.POSToken`)
U-Compare type details: In D2.2, it was planned that this component would originally take annotations of type `org.u_compare.shared.syntactic.Token`. However, as described in the LX-Tokenizer entry above, this annotation type cannot store all the information required as input to the LX-Tagger, hence the creation of the `pt.ul.fc.di.nlx.LXToken` annotation type. Since this annotation type is an extension of `pt.ul.fc.di.nlx.LXToken`, the output annotation type can remain the same as the input type – all that is required is to set the value of the `posString` attribute, to record the part-of-speech tag assigned by the LX-Tagger.
Implementation details/issues: The tool is implemented as a web service, LXService. In order to implement the U-Compare component, the web service is invoked through a package, available as lxServiceClient.jar, which must be placed in the classpath of U-Compare. Its constructor requires one parameter related to the authentication of the client, namely the client's username, as this is registered at the LXService database of clients. We have created a new user in this database for U-Compare. The output of the call to web service is then converted to the format required by U-Compare.

LX-Tagger

Description: An alternative part–of-speech tagger for Portuguese, which operates on plain, unannotated text. Tokenization and part-of-speech tagging are both performed by this tool.
Languages covered: Portuguese
Original resource implementation: Web service
Input: Plain text
U-Compare input type: N/A
Output: Tokenised text with parts-of-speech assigned
U-Compare output type:
`org.u_compare.shared.syntactic.POSToken`
U-Compare type details: The input and output types belong to the U0-Compare type system, as originally envisaged in D2.2
Implementation details/issues: The tool is implemented as a web service, LXService. In order to implement the U-Compare component, the web service is invoked through a package, available as lxServiceClient.jar, which must be placed in the classpath of U-Compare. Its constructor requires one parameter related to the authentication of the client, namely the client's username, as this is registered at the LXService database of clients. We have created a new user in this database for U-Compare. The output of the call to web service is then converted to the format required by U-Compare.

4.1.2 Corpora

CINTIL Corpus

Description: This component provides excerpts from CINTIL-Corpus Internacional do Português. This corpus is a linguistically interpreted corpus of Portuguese. Excerpts are composed of 15,992 annotated tokens, each one of which is verified by human expert annotators.
Languages covered: Portuguese
Annotation types: Sentences, POS tags
U-Compare annotation types:
org.u_compare.shared.syntactic.POSToken
org.u_compare.shared.syntactic.Sentence
U-Compare type details: The annotation types are the same ones originally planned in D2.2. They are types belonging to the original U-Compare type system.
Implementation details/issues: No issues encountered – straightforward implementation of UIMA reader for the corpus

4.2 IST – Instituto Superior Técnico

E-txt2db components

Description: In contrast to all other tools listed, the tool provided is not a UIMA component itself, but rather automatically generates named entity tagger UIMA components through training using the E-txt2db Component Generator
Languages covered: Any (according to the training data)
Original resource implementation language: Java
Input: Depends on the component, either plain text or text with annotations already added.
U-Compare input type: The E-txt2db Component Generator allows for the generation of named entity tagger UIMA components that receive plain text as input. However, it also allows for the generation of named entity taggers that rely on other annotations (e.g. Token can be used to perform tokenization, Sentence can be used to split the text into sentences, POSToken can be used as a feature to help in the tagger's decision).
Output: Named entity annotations, according to observations in the training data
U-Compare output type: `etxt2db.annotators.NamedEntityTyped` (extends `org.u_compare.shared.semantic.NamedEntity`)
The components produce named entities according to what was observed in the training data.
U-Compare type details: The output type produced is not the originally-planned U-Compare type (`org.u_compare.shared.semantic.NamedEntity`), because that type of annotation does not contain a field corresponding to the tag of entity produced. Since the tool can be trained to recognize a potentially infinite set of named entity types, it was not a practical solution to create new types for each different type of named entity. Instead, the type `etxt2db.annotators.NamedEntityTyped` was created, which provides a string-valued field to store the type of the named entity recognised.
Implementation details/issues: Creating the code for the generated components themselves was straightforward. The challenge was to produce the components automatically through training (E-txt2db Component Generator). The solution used produce components automatically was to develop methods to produce XML files corresponding to the descriptor of the component that uses the model that is produced by E-txt2db during the training phase.

4.3 University of Manchester – UNIMAN

4.3.1 Tools

Apertium Morphological Analyser (Updated from D4.4)

Description: Tokenises text and assigns one or more possible part-of-speech-tags/morphological analyses to each token.
Languages covered: English, Portuguese, Spanish, Catalan, Galician, Basque
Original resource implementation: Java port of original C++ code.
Input: Plain text
U-Compare input type: N/A
Output: One or more morphological analyses for each token, consisting of part-of-speech tags, together with morphological information, such as base form, person, number and gender.
U-Compare output type: `org.u_compare.shared.syntactic.ApertiumToken` [subtype of `org.u_compare.shared.syntactic.POSToken`]
U-Compare type details: The type `org.u_compare.shared.syntactic.POSToken` has been extended to allow the additional morphological information produced by the Apertium morphological analyser to be stored as part of the annotation.
Implementation details/issues: This is one module of the Apertium machine translation system (third-party, open-source software). The version of the wrapper delivered as part of D4.4 suffered from robustness issues, such as crashing when blank lines or slashes were encountered in the input text. The component has been more thoroughly tested, and these and other bugs have been eliminated. In addition, the use of the component has been made more straightforward for users. Instead of having to provide a path to linguistic data file, the data files for the languages relevant to the project are included within the component jar file. All the user now has to do is to specify an appropriate language pair as a parameter. The possible values are "en-es", "es-en", "gl-es", "es-gl", "es-pt", "pt-es", "es-ca", "ca-es", "es-ro" and "eu-es". The reason for specifying a language pair is that this is a module of the Apertium machine translation system. The morphological analyser can, however, be used independently of the translation module. In this case, the code for the language to be analysed must be the first one in the pair. The original version of the component specified Romanian as a possible language. However, the quality of the output was found to be poor, and so this language has been removed.

Apertium Tagger (Updated from D4.4)

Description: Determines the most appropriate part-of-speech tag/morphological analysis for each token in a text, from amongst possible analyses output by the Apertium morphological analyser module (see above)
Languages covered: English, Portuguese, Spanish, Catalan, Galician, Basque, Romanian
Original resource implementation: Java port of original C++ code.
Input: Morphologically analysed text (analysis MUST be produced by running the Apertium Morphological Analyser prior to this component)
U-Compare input type: `org.u_compare.shared.syntactic.ApertiumToken` [subtype of `org.u_compare.shared.syntactic.POSToken`]
Output: One or more morphological analyses for each token, consisting of part-of-speech tags, together with morphological information, such as base form, person, number and gender.
U-Compare output type: `org.u_compare.shared.syntactic.ApertiumToken` [subtype of `org.u_compare.shared.syntactic.POSToken`]
U-Compare type details: The type `org.u_compare.shared.syntactic.POSToken` has been extended to allow morphological information to be stored about tokens.
Implementation details/issues: This is one module of the Apertium machine translation system (third-party, open-source software). The component wrapper delivered with D4.4 has been improved in a couple of ways. Firstly, the information stored in the output annotations has been stored to make in more understandable/readable by users. Rather than storing the "raw" information output by Apertium within these annotations, which combines base forms, morphology and POS tags in a specific format, this information has been separated and is stored separately as the values of the "base", "morphology" and "POSString" attributes of the `ApertiumToken` annotations, respectively. In addition, the use of the component has been made more straightforward for users. Instead of having to provide a path to linguistic data file, the data files for the languages relevant to the project are included within the component jar file. All the user now has to do is to specify an appropriate language pair as a parameter. The possible values are "en-es", "es-en", "gl-es", "es-gl", "es-pt", "pt-es", "es-ca", "ca-es", "es-ro" and "eu-es". The reason for specifying a language pair is that this is a module of the Apertium machine translation system. The tagger can, however, be used independently of the translation module. In this case, the code for the language to be analysed must be the first one in the pair. The original version of the component specified Romanian as a possible language.

However, the quality of the output was found to be poor, and so this language has been removed.

Apertium Translator

Description: Given morphologically analysed and POS tagged text, outputs translated text with morphological analyses and POS tags attached.
Languages covered: English, Portuguese, Spanish, Catalan, Galician, Basque.
Original resource implementation: Java port of original C++ code.
Input: Morphologically analysed and POS tagged text (MUST be produced by running the Apertium Morphological Analyser and Apertium tagger in a pipeline, prior to this component)
U-Compare input type: `org.u_compare.shared.syntactic.ApertiumToken` [subtype of `org.u_compare.shared.syntactic.POSToken`]
Output: Alternative view of the text (second "sofa" (subject of analysis) in UIMA parlance), containing translated text with morphological analyses and POS tags attached.
U-Compare output type:
`org.u_compare.shared.syntactic.ApertiumToken` [subtype of `org.u_compare.shared.syntactic.POSToken`] in the `targetSofa` view of the text.
U-Compare type details: The type `org.u_compare.shared.syntactic.POSToken` has been extended to allow the additional morphological information to be stored about tokens in the source and target language text.
Implementation details/issues: This component is provided instead of the separate "Apertium MT Transfer" and "Apertium Morphological Generator" components that were originally planned in D2.2. The plan was for the MT transfer component to generate base forms in the target language with morphological information attached, whilst the morphological generator would generate appropriate inflected forms. However, this turned out not to be practical, as it is problematic to change the document text in a sofa once it has been set (i.e., to change base forms to inflected forms). Thus, the components were combined, so that the correctly inflected text is set as document text in the targetSofa, "ApertiumToken" annotations are created for each token in the targetSofa, and morphological and POS tags are set as attributes of the annotation. Using the current version of U-Compare, this component can be run as part of a workflow, but the targetSofa cannot yet be viewed (this extension to U-Compare is planned during the final 6 months of the project). However, at present, the targetSofa can be viewed by saving the

output of the workflow to an XMI file (by adding an “Xmi Writer CAS consumer” to the end of the workflow). Then, the XMI file can be viewed using the UIMA Annotation Viewer, which supports viewing of mulitiple sofas. The Apertium Translator component requires a language pair to be specified as a parameter. The possible values are "en-es", "es-en", "gl-es", "es-gl", "es-pt", "pt-es", "es-ca", "ca-es" and "eu-es".

Cafetiere Sentence Splitter (New component not mentioned in D2.2)

Description: Given plain text, outputs sentence boundaries
Languages covered: English
Original resource implementation: Java
Input: Plain text
U-Compare input type: `N/A`
Output: Text with sentence boundaries marked
U-Compare output type: `org.u_compare.shared.syntactic.Sentence`
U-Compare type details: This simple component output annotation belonging to an existing U-Compare type.
Implementation details/issues: Straightforward wrapping of a simple sentence splitting tool that was already implemented in Java.
NOTE: This component replaces the NaCTeM sentence breaker in workflows, due to a potential licensing issue with the NaCTeM sentence breaker.

4.3.2 Corpora

GENIA Corpus

Description: 1,999 Medline abstracts in the biomedical domain annoted with various types of linguistic information
Languages covered: English
Annotation types: POS tags, Named Entities, co-reference, treebank annotations
U-Compare annotation types:
org.u_compare.shared.syntactic.POSToken,
org.u_compare.shared.semantic.NamedEntity,
org.u_compare.shared.semantic.Constituent,
org.u_compare.shared.semantic.CoreferenceAnnotation
U-Compare type details: The annotation types are the same ones originally planned in D2.2. They are types belonging to the original U-Compare type system.
Implementation details/issues: No issues encountered – straightforward implementation of UIMA reader for the corpus

GENIA Event Corpus

Description: 1000 abstracts of the GENIA corpus. It contains 9,372 sentences in which 36,114 events are identified. New layers of annotation (meta-knowledge) have been added as part of the METANET4U project to allow advanced information extraction involving opinion mining, inconsistency and contradiction checking, entailment and hedging.
Languages covered: English
Annotation types: Named Entities, Biomedical events, Meta-knowledge annotation
U-Compare annotation types:
org.u_compare.shared.semantic.NamedEntity,
jp.ac.u_tokyo.s.is.www_tsujii.corpus.genia.GeniaEventAnnotation
[subtype of org.u_compare.shared.semantic.EventAnnotation],
org.u_compare.shared.semantic.MetaKnowledgeEvent
[subtype of org.u_compare.shared.semantic.EventAnnotation]
U-Compare type details: The annotation types are the same ones originally planned in D2.2. They are types belonging to the original U-Compare type system.
Implementation details/issues: No issues encountered – straightforward implementation of UIMA reader for the corpus

GREC Corpus

Description: 240 MEDLINE abstracts annotated with gene regulation events arranged around verbs and nominalised verbs. A rich set of 13 semantic role types are used to characterize the event arguments. Named entities within event arguments are also identified.
Languages covered: English
Annotation types: Named Entities, Biomedical events, Meta-knowledge
U-Compare annotation types:
org.u_compare.shared.semantic.NamedEntity,
org.u_compare.shared.semantic.EventAnnotation,
org.u_compare.shared.semantic.MetaKnowledgeEvent
[subtype of org.u_compare.shared.semantic.EventAnnotation]
U-Compare type details: The annotation types are the same ones originally planned in D2.2. They are types belonging to the original U-Compare type system.
Implementation details/issues: No issues encountered – straightforward implementation of UIMA reader for the corpus

4.4 University Alexandru Ioan Cuza (UAIC)

4.4.1 Tools

UAIC POS tagger (New component not mentioned in D2.2)

Description: This tool integrates a sentence splitter a tokenizer and a pos tagger. It takes as input raw text.
Languages covered: Romanian, English
Input: Plain text
U-Compare input type: N/A
Output: sentences tokens and pos taggs
U-Compare output type: `org.u_compare.shared.syntactic.Sentence` `RichToken`
U-Compare type details: All output types are U-Compare types
Implementation details/issues: The reason for the integration of the sentence splitter, tokenizer and pos tagger as one tool is that the internal components share a huge dictionary. It is recommended to use this tagger in conjunction with other UAIC tools, because the tagset used by this tagger is not 100% compatible with RACAI's pos tagger. Many tools have never been tested with RACAI's tagger, or have been trained using output from UAIC POS tagger.

Splitter-UAIC v1

Description: Identifies segments in sentences. For example "John saw, although he didn't expect, that he had a lot of money/.", would be split into 3 segments, "John saw", "although he didn't expect" and "that he had a lot of money".
Languages covered: Romanian, English
Input: Plain text
U-Compare input type: N/A
Output: segments, as explained above
U-Compare output type: `org.u_compare.shared.document.Fragment`
U-Compare type details: It was originally thought that a new U-Compare type may be needed. However, closer examination of the U-Compare type system revealed that a suitable type existed.

Implementation details/issues: No problems encountered - straightforward wrapping, given compatibility between original implementation language and UIMA.

Splitter-UAIC v2 (UaicClauseSplitter) (New component not mentioned in D2.2)

Description: Identifies segments in sentences. For example "John saw, although he didn't expect, that he had a lot of money/.", would be split into 3 segments, "John saw", "although he didn't expect" and "that he had a lot of money".
Languages covered: Romanian
Input: Sentences, pos tagged tokens, and NP chunks (optional)
U-Compare input type: `org.u_compare.syntactic.Sentence, org.u_compare.syntactic.RichToken, uaic.uimatypes.NpChunkWithHead(optional)`
Output: segments, as explained above
U-Compare output type: `uaic.uimatypes.Clause`
U-Compare type details: Since the Clause type is more relevant for a node structure rather than a sentential clause annotation, the decision was to make another uaic type.
Implementation details/issues: Both variants (for English and Romanian) of the clause splitter have been trained to recognise clause boundaries from manually annotated gold corpora. The model uses as features: pos-tags and discourse markers. A clause boundary could not reside in-between the limits of a NP. The English variant is currently undergoing testing; the component will be updated to provide functionalty for English prior to the end of the project.

FDG-parser-UAIC

Description: Functional Dependency Grammar parser
Languages covered: Romanian
Input: POS-tagged text and sentence markers
U-Compare input type: `org.u_compare.shared.syntactic.Sentence, org.u_compare.shared.syntactic.RichToken`
Output: FDG dependencies between words belonging to the same sentence
U-Compare output type: `org.u_compare.shared.syntactic.ConllDependency`
U-Compare type details: In D2.2, it was stated that a subtype of `org.u_compare.shared.syntactic.Dependency` may be required to encode the output of this component. Since this parser was developed for the CoNLL shared task, the subtype

`org.u_compare.shared.syntactic.ConllDependency` has been used, which is already part of the U-Compare type system. The output type also contains the start and end features from Annotation but they are not to be used.
Implementation details/issues: In D2.2, it was erroneously stated that this component operates on both English and Romanian, but actually it only operates on Romanian.

NP-chunker-UAIC

Description: Detects noun phrases
Languages covered: Romanian
Input: POS-tagged text
U-Compare input type: `org.u_compare.shared.syntactic.RichToken`
Output: NP chunks
U-Compare output type: `uaic.uimatypes.NpChunkWithHead`
U-Compare type details: The output type is an extension of `org.u_compare.shared.syntactic.chunk`, which allows the head of each NP to be stored.
Implementation details/issues: In D2.2, it was erroneously stated that this component requires FDG parse results, in addition to POS-tagged output. Whilst it was originally envisaged that this component would operate on both Romanian and English, it currently operates only on Romanian.

Summarizer-UAIC

Description: Provides a summary of text
Languages covered: Romanian, English
Input: Sentence segments
U-Compare input type: `org.u_compare.shared.document.Fragment`
Output: A summary of the text
U-Compare output type: No output annotations as such; the output summary is stored as an alternative view of the text, i.e., in a second sofa called *Summary.*
U-Compare type details: No new types needed to be defined for this component, given that the multiple sofa approach was used, rather than defining a new type to hold the summary as an annotation in the original sofa.
Implementation details/issues: The solution used by this component to store the summary is similar to the one chosen to store text that is automatically translated by the *Apertium Translator* component, as described above. As was also explained above, the current version of U-Compare allows components that produce multiple text views to be run, but the output of new sofas created by the component cannot yet be

visualized. Whilst an extension to U-Compare to allow this multiple sofas to be viewed using the interface is planned to be completed before the end of the project, at present, an alternative solution must be used to verify the output of this component. The output of the workflow can be saved to an XMI file by adding an "Xmi Writer CAS consumer" to the end of the workflow). Then, the XMI file can be viewed using the UIMA Annotation Viewer, which supports viewing of multiple sofas.
NOTE: A more complex version of the summarizer, which makes use of the output of DP-UAIC (see below) will be provided prior to the end of the project.

DP-UAIC

Description: Discourse Parser. Produces discourse trees for an input text.
Languages covered: Romanian
Input: Sentences, Tokens, POS tags, clause phrases (output of Splitter-Uaic v2), NP chunks (optional), co-reference chains (optional; output of RARE-UAIC)
U-Compare input type:
`org.u_compare.shared.syntactic.Sentence`**,**
`org.u_compare.shared.syntactic.Token`
`org.u_compare.shared.semantic.CoreferenceAnnotation,`
`uaic.uimatypes.Clause,`
`uaic.uimatypes.NpChunkWithHead` (Optional)
`org.u_compare.shared.semantic.CoreferenceAnnotation` (Optional)
Output: Discourse trees
U-Compare output type: `uaic.uimatypes.DiscourseNode`
U-Compare type details: The output is represented as a binary tree of DiscourseNode annotation types. A suitable type did not exist in the existing U-Compare type system, and so this new type was added.
Implementation details/issues: The nodes which do not have children (leaf nodes) resemble a clause (which is accessible through the Clause feature). Each specifies its parent vein and its heads. A version of this component that can also handle English will be provided prior to the end of the project.

4.5 RACAI – Romanian Academy

4.5.1 Tools

RACAI Language Identifier

Description: This is a UIMA, U-Compare compatible wrapper to the RACAI's language identifier (LangID) web service hosted at http://www.racai.ro/webservices/LangId.asmx. LangID is able to identify 54 commonly known languages (*Afrikaans*, *Alemannic German*, *Arabic*, *Azerbaijani*, *Bavarian*, *Belarusian*, *Bosnian*, *Breton*, *Bulgarian*, *Catalan*, *Chinese* (*Standard*), *Croatian*, *Czech*, *Danish*, *Dutch*, *English*, *Esperanto*, *Estonian*, *Filipino*, *Finnish*, *French*, *Galician*, *German*, *Greek*, *Hebrew*, *Hungarian*, *Indonesian*, *Irish Gaelic*, *Italian*, *Japanese*, *Korean*, *Latin*, *Latvian*, *Lithuanian*, *Maltese*, *Norwegian*, *Occitan*, *Polish*, *Portuguese*, *Romanian*, *Russian*, *Serbian*, *Serbo-Croatian*, *Sicilian*, *Slovak*, *Slovene*, *Spanish*, *Swedish*, *Thai*, *Turkish*, *Ukrainian*, *Volapük*, *Welsh*, *Yiddish*) and 3 rare languages (*Aweti*, *Beaver*, *Teop*). However, the U-Compare component is hardcoded to not recognize the rare languages.
Original resource implementation language: C# under Microsoft .NET
Input: Input data must be UTF-8 encoded text, which should be large enough to ensure correct language detection. The recommended size is around a couple of substantial paragraphs.
U-Compare input type: None. The LangID U-Compare component simply gets the text of the input document (via JCas.getDocumentText()) which it passes to the web service for language recognition.
Output: The language code conforming to ISO 639-1 or the value 'uncertain' if the confidence threshold was too low (smaller than 0.7, hardcoded value). This can happen if the input text is too short and/or the languages are alike (e.g. Croatian vs. Slovene vs. Serbian).
U-Compare output type: None. The U-Compare component simply sets the recognized language on the input document via JCas.setDocumentLanguage().
U-Compare type details: No U-Compare types were used.
Implementation details/issues: Straightforward UIMA wrapper around the web service. The LangID component can be added to the beginning of a workflow consisting of the other TTL components that were provided in D2.2 (Tokenizer, Tagger, Lemmatizer, Chunker) in order to determine the correct configuration of the components to use (Romanian, English or French) without having to set this manually. The LangID component could also be used for other types of components that can handle more than one language.

4.6 University of Malta (UOM)

4.6.1 Tools

MLRS Part-of-speech tagger
Description: Assigns parts-of-speech, given normal text
Languages covered: Maltese
Original resource implementation: Java application
Input: Plain text
U-Compare input type: Plain text
Output: Tokens with parts of speech assigned

U-Compare output type:
org.u_compare.shared.syntactic.POSToken.
U-Compare type details: This is originally planned output type. The existing U-compare type is sufficient for the output of this tool, without any need for extension.
Implementation details/issues:
TNT, a third-party software is used for training and tagging the text. This software imposes some restrictions on the execution environment, namely:
- it has to be run under Linux, and
- as input, it takes text file with one word per line

We eventually decided that this component be implemented, and made available through a web-service.

This was decided after several other attempts:

1st Attempt: We first started with a small program which (i) takes an input (text), (ii) formats it for TNT, (iii) executes the tagger using bash commands, and (iv) reads back the output. After testing this, it was then wrapped as a U-Compare component. Testing this component through Eclipse worked; but when exporting this as a standalone component (.jar) to be included in the U-Compare library, the component failed to execute.

2nd Attempt: Assuming that the 1st attempt failed because of insufficient permissions, another attempt was made. A web-service was implemented which takes text as input and return its tagged equivalent. The U-Compare component then reads the text from the CAS, calls the web-service, and adds the results back to the CAS. The component was then first tested through Eclipse. Similarly to the first attempt, the component worked when tested through Eclipse, but failed to work when tested as a standalone component.

3rd Attempt: Assuming that the 2nd attempt also failed due to insufficient permissions, we decided to adopt the 1st attempt's implementation, but have the whole component running on a server and deployed as a web-service (using a deployment script that comes with UIMA). This way, the component would have the necessary permissions to execute. This was then tested through a client Java application, which makes use of the component and everything seemed to work as expected.

4.7 UPC - Universitat Politècnica de Catalunya

UPC is not delivering any new components at this point. The Ogimos text-to-speech tool is planned for delivery as a UIMA component during the lifetime of the project. However, the development of this component is dependent on new functionality being added to U-compare that will allow speech files to be played. Since this enhancement to U-Compare is planned during the coming months, Ogimos will be implemented as a UIMA component when the new U-Compare functionality is available.

4.8 UPF- Universitat Pompeu Fabra

The main involvement of UPF is in the PANACEA workflow system. However, the plan has been for UNIMAN to work on making available some of their PANACEA web services as UIMA components, as time permits, since these provide a basic set of processing tools for the Spanish and Catalan languages, including tokenization, morphological analysis, tagging and parsing. A total of 8 potential web services were identified in Deliverable 2.2. Of these, 2 have currently been implemented as web services by UNIMAN. The implementation of the remaining services as UIMA components should be more straightforward, and implementation of these is planned during the final phase of the project.

<u>Iula_tokenizer</u>
Description: Tokenizes plain text. Also identifies sentence and paragraph boundaries, and recognizes proper names.
Languages covered: Spanish, Catalan
Original resource implementation: Java application
Input: Plain text
U-Compare input type: `N/A`
Output: Paragraphs, sentences, tokens are named entities.
U-Compare output type:
```
org.u_compare.shared.document.text.Paragraph
org.u_compare.shared.syntactic.Sentence
org.u_compare.shared.syntactic.Token
org.u_compare.shared.semantic.NamedEntity
```
U-Compare type details: Only Token annotations were originally planned as output annotations for this component in D2.2. However, at

that point, the information about the web service output had been obtained by looking only at the documentation for the service. Subsequent experimentation with the service has revealed that a greater range of annotations are output than originally anticipated, hence the increased range of U-Compare output annotation types.
Implementation details/issues: No specific issues encountered.

Iula_tagger
Description: Tokenizes plain text and assigns part of speech tags. Also identifies sentence and paragraph boundaries, and recognizes proper names.
Languages covered: Spanish, Catalan
Original resource implementation: Java application
Input: Plain text
U-Compare input type: `N/A`
Output: Paragraphs, sentences, tokens are named entities.
U-Compare output type:
```
org.u_compare.shared.document.text.Paragraph
org.u_compare.shared.syntactic.Sentence
org.u_compare.shared.syntactic.POSToken
org.u_compare.shared.semantic.NamedEntity
```
U-Compare type details: Only POSToken annotations were originally planned as output annotations for this component in D2.2. However, at that point, the information about the web service output had been obtained by looking only at the documentation for the service. Subsequent experimentation with the service has revealed that a greater range of annotations are output than originally anticipated, hence the increased range of U-Compare output annotation types.
Implementation details/issues: No specific issues encountered.

5 Workflows delivered or updated at M18

The resources that have been wrapped as U-Compare components, as detailed in the last section, can currently be combined together into workflows to perform 18 out of the 26 NLP tasks that were specified in Deliverable 2.2. Several of these workflows can operate on multiple languages and can use various combinations of components developed by different partners. The 20 new UIMA components available as part of this deliverable have added 8 additional workflows to those that could be implemented with the components delivered as part of deliverable D4.4. The new workflows generally constitute more sophisticated language processing tasks, or evaluation workflows that are possible due to the new availability of gold standard annotated corpora as UIMA components. In addition, certain workflows that could already be implemented at M12 can now be implemented using alternative components and/or they operate in a greater number of languages.

For most workflows, a potentially large number of "paths" through the workflow are possible, given that multiple components can be substituted at each step. In D2.2, a set of conceptual diagrams were provided, which illustrated all possible perceived paths though these 26 tasks, using the components that were planned to be made available during the METANET4U project. The diagrams showed which components could be used to move from one state of the workflow to the next (e.g., to move from sentence annotations to token annotations) for different languages. Therefore, for each task, a potentially very large number of workflows would be possible by choosing different possible components at each stage of the workflow.

As has been illustrated above, workflows can be constructed easily in U-Compare by dragging components from the library onto the workflow canvas, in a particular order. Therefore, the diagrams can be used as a guide to the possible workflows that could be built using the components that are being made available in METANET4U.

Particular workflows can be exported from U-Compare as single files ("ucz" or U-Compare zip files). As part of this deliverable, we provide a set of sample "implemented" workflows as ucz files, which can be imported into U-Compare, to complement those already provided as part of D4.4. The new workflows provided as part of this deliverable make use of one of more components that have been wrapped as UIMA components since D4.4. As was the case for D4.4, given the large number of potential alternative workflows that can be created for each NLP task, we do not provide an exhaustive set of all possible implemented workflows. Rather, we include a selection of sample implemented workflows, consisting of at least one workflow for each task-language pair (e.g. *tokenisation-Maltese*). The sample workflows are currently available on the METANET4U intranet: http://metanet4u.eu/intranet/index.php/WPS_-_Webservices_%28coord:_Sophia%29#Sample_workflows

The aim of this sample set is to provide a set of easily importable and immediately usable workflows for different tasks and languages. These workflows can act as templates that the user can subsequently, change, extend or configure, by substituting different components. The idea is that such templates make it easier for users to experiment with different configurations of workflows, than having to build them from scratch. Where possible, each sample workflow combines components developed by different partners, in order to highlight the ease of interoperability that can be achieved through the use of UIMA wrapping and U-Compare.

On the following pages, conceptual diagrams are shown of the 18 workflows for which at least one of the potential paths can be constructed, using the UIMA components that are currently available. The format of these diagrams is the same as those shown in D4.4, but the description of the diagrams is again provided below.

The circles in the diagrams represent the possible different information states that can occur between the input information state and the output information state. Lines represent the possible ways to move between the information states. Each line is labelled with the individual components that can be used to produce the information to move between one state and the next. For example, a part-of-speech tagger can be used to move from the "token" state to the "POS" state. Each resource is represented in the diagrams as a number, with a full description in the legend, as follows:

<partner_short_name>:<tool_name>:<languages_covered>

For example, *ULX:Chunker:pt*, represents the chunker tool developed by the University of Lisbon, which works on the Portuguese language.

Each workflow diagram shows all possible tools that can be used to carry out each step of the processing in all of the available languages, including those that have not yet been implemented as U-Compare components. Only if the complete workflow can be carried out for a particular language are the tools for that language displayed in the diagram. For example, lemmatization is not possible for Portuguese, and so no Portuguese tools are shown in the lemmatization workflow, even though most of the intermediate steps can be carried out using Portuguese tools. This makes it straightforward to determine exactly which types of workflow are currently possible for each language.

In the diagrams, some lines skip individual states. This is because of the different processing capabilities of different tools. For example, some part-of-speech taggers may require tokenized text as input, whilst other taggers may operate directly on plain text, and perform tokenization as an integral part of the tool.

5.1 Augmented workflows

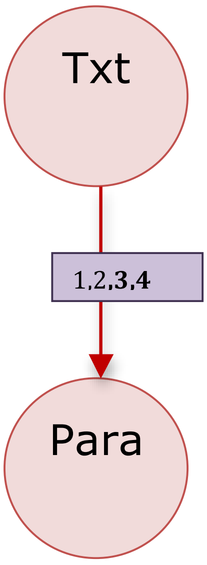
The diagrams in this section correspond to workflows that could already be partially implemented in D4.4, but for which certain languages or paths through the workflow were not possible at that time. The diagrams provided here indicate how workflows are more fully implementable now that the extra UIMA components have been made available. The features of the diagrams are as follows:

- Resources that have already been wrapped are shown in black text.

- Any new components that have been made available since D4.4 are indicated using italics.
- Resources that have not yet been wrapped as UIMA components are shown using red text.
- Grey arrows indicate which of the planned transitions between information states that are not currently possible at all (due to a lack of available wrapped components).
- Languages that were planned for a particular workflow, but which are not currently possible are shown in red.
- Any newly supported languages since D4.4 are shown using italics. Languages that are newly supported through the availability of new components are underlined.
- Any changes to planned components or their usage since D4.4 are indicated using bold font. These changes may include extra components that were not originally planned, or else a change in the planned usage of the component (i.e., at a different place in the workflow).

Paragraph breaking

Purpose: Identifies paragraphs in plain text
Languages: Any



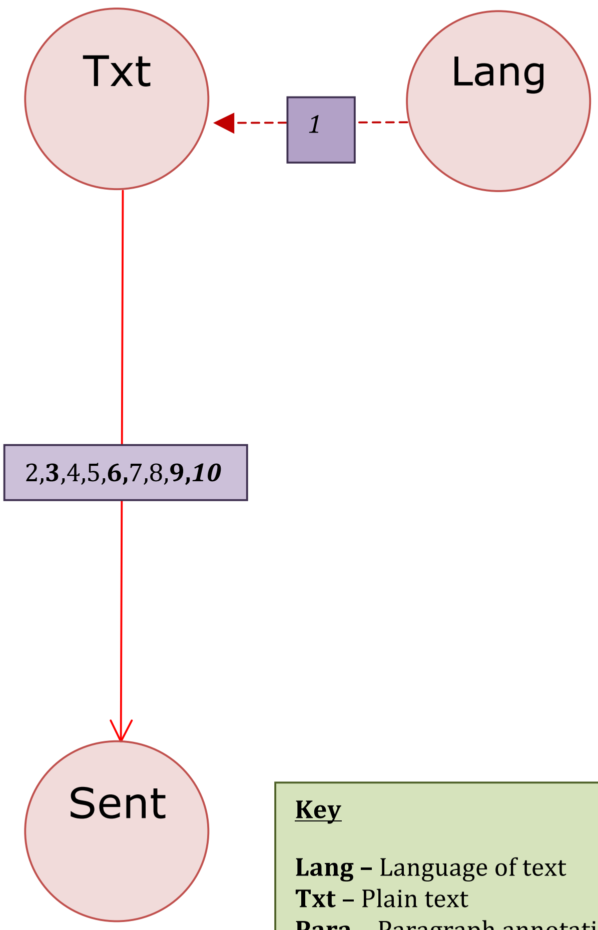
Key

Txt – Plain text
Para – Paragraph annotations

Resources

1 - ULX: Chunker: pt
2 - UOM: MLRS Paragraph breaker: All
3 – UPF/UNIMAN: IULA_tokenizer:es,ca
4 – UPF/UNIMAN: IULA_tagger:es,ca

Sentence splitting
Purpose: Identifies individual sentences in plain text
Languages: All



<u>Resources</u>

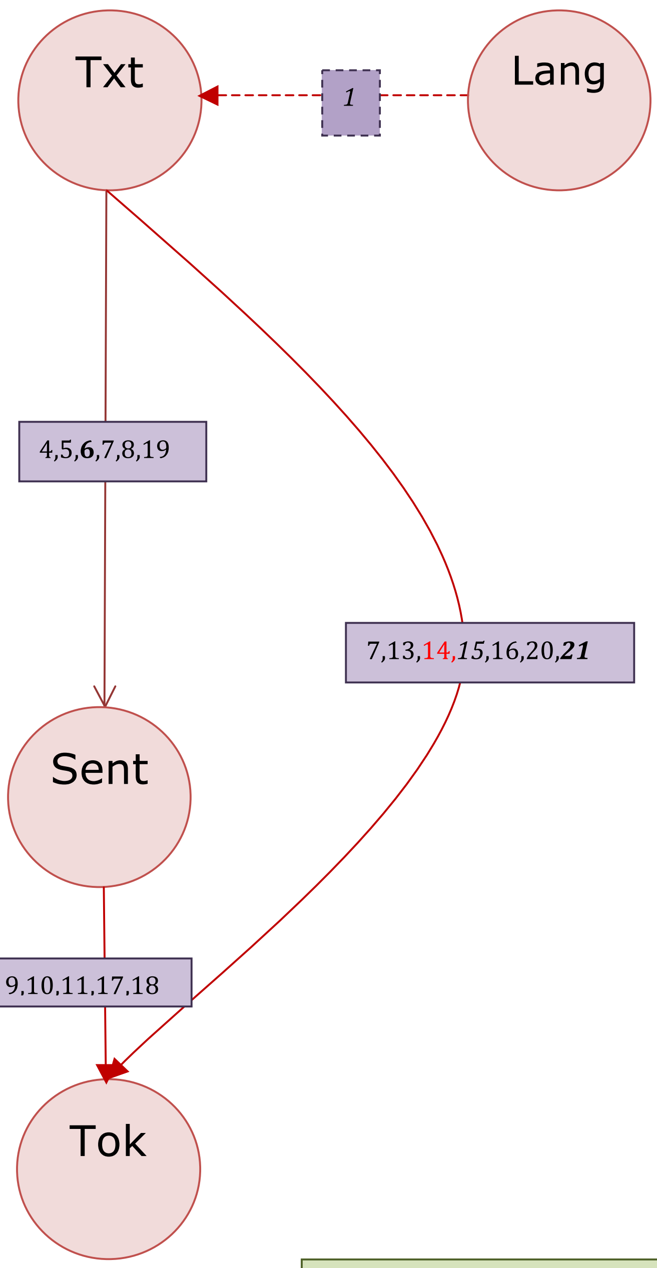
1 - RACAI:Lang Identifier
2 - ULX: Chunker: pt
3 –UPF/UNIMAN: IULA_tokenizer:es,ca
4 - UNIMAN: GENIA sentence splitter:en
5 - UNIMAN: OpenNLP sentence detector: en
6 - UNIMAN: Cafetiere Sentence Splitter: en
7 - RACAI:TTL-Tokenizer:ro,en,fr
8 - UOM: Sentence Splitter: Any
9 – UPF/UNIMAN: IULA_tagger:es,ca
10 – UAIC: PosTagger-UAIC:ro

<u>Key</u>

Lang – Language of text
Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations

Tokenization

Purpose: Identifies individual tokens in plain text
Languages: Any



Resources

1 - RACAI:Lang Identifier
4 -UNIMAN:Genia Sentence Splitter: en
5- UNIMAN:OpenNLP sentence detector: en
6 - UNIMAN:Cafetiere sentence splitter: en
7- RACAI:TTL-Tokenizer:ro,en,fr
8 - ULX: Chunker: pt
9 - UNIMAN:Genia Tagger (with tokenization): en
10 - UNIMAN:Stepp Tagger (with tokenization): en
11 - UNIMAN:OpenNLP tokenizer:en
13 - UAIC: TokenizerUAIC: Any
14 -UPF: freeling_tokenizer: es,ca
15 -UPF: iula_tokenizer: es,ca
16 -UNIMAN: Apertium Morpho Analyser: en,es,ca,pt,gl,eu
17 -UOM: MLRS Tokenizer:mt
18 -ULX: Tokenizer:pt
19 – UOM: MLRS Sentence Splitter:Any
20- UOM: MLRS Tokenizer:mt
21 – UAIC: PosTagger-UAIC:ro

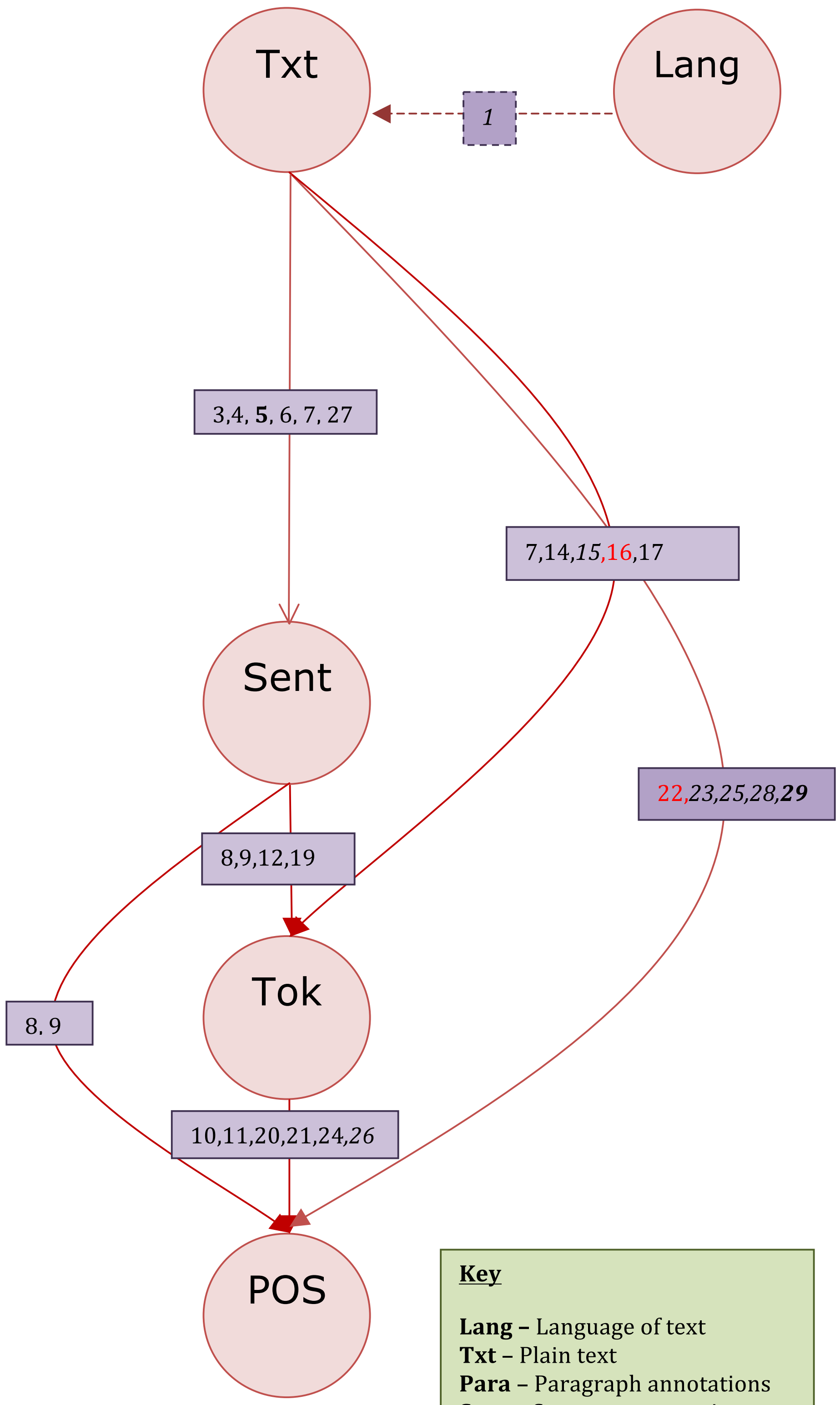
Key

Lang – Language of text
Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations

Part-of-speech tagging

Purpose: Identifies individual tokens in plain text and assigns parts-of-speech to them

Languages: En, Es, Ca, Pt, Gl, Eu, Ro, Fr, *Mt*



Resources

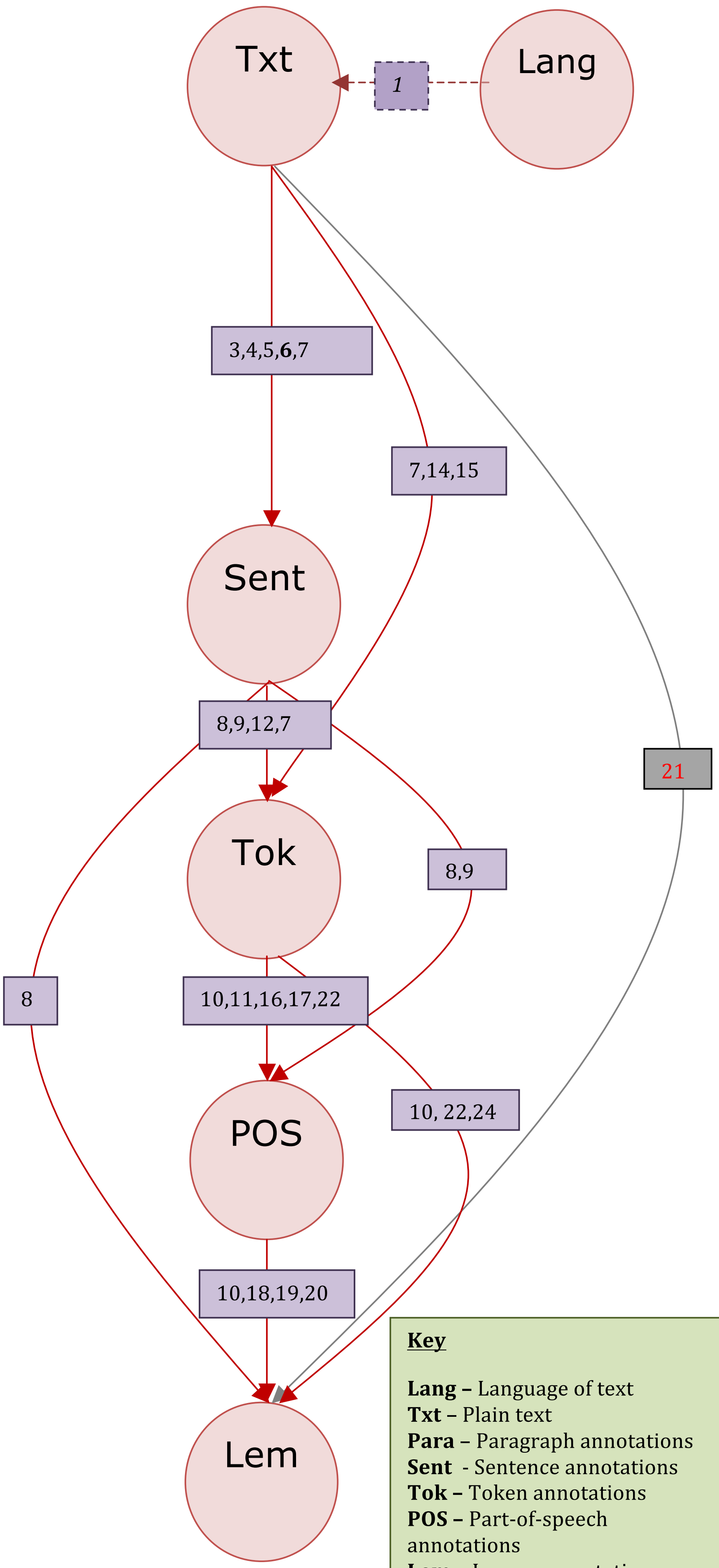
1 - RACAI:Lang Identifier
3 - UOM:MLRS Sentence Splitter:Any
4 - UNIMAN:Genia Sentence Splitter: en
5- UNIMAN: Cafetiere sentence spllitter: en
6 - UNIMAN:NaCTeM sentence breaker:en
7- RACAI:TTL-Tokenizer:ro,en,fr
8 - UNIMAN:Genia Tagger (with tokenization): en
9 - UNIMAN:Stepp Tagger (with tokenization): en
10 - UNIMAN:Genia Tagger (no tokenization): en
11 - UNIMAN:Stepp Tagger (no tokenization): en
12 - UNIMAN:OpenNLP tokenizer:en
14 - UAIC: TokenizerUAIC: Any
15 - UPF: iula_tokenizer: es,ca
16 - UPF: freeling_tokenizer: es,ca
17 - UNIMAN: Apertium Morpho Analyser:en,es,ca,pt,gl,eu
19 - ULX: Tokenizer:pt
20 - UNIMAN:OpenNLP Tagger:en
21 - RACAI:TTL Tagger:ro,en,fr
22 -UPF: freeling_tagging: es,ca
23 -UPF: iula_tagger: es,ca
24 - UNIMAN: Apertium Tagger: en,es,ca,pt,gl,eu
25 - UOM:POS Tagger:mt
26 - ULX: LX-POSTagger:pt
27- ULX:Chunker:pt
28 – ULX: LX-Tagger:pt
29 – UAIC: PosTagger-UAIC:ro

Key

Lang – Language of text
Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations
POS – Part-of-speech annotations

Lemmatization

Purpose: Identifies individual tokens in plain text and assigns lemma information to them
Languages: En, Es, Ca, Pt, Gl, Eu, Ro, Fr.



Resources

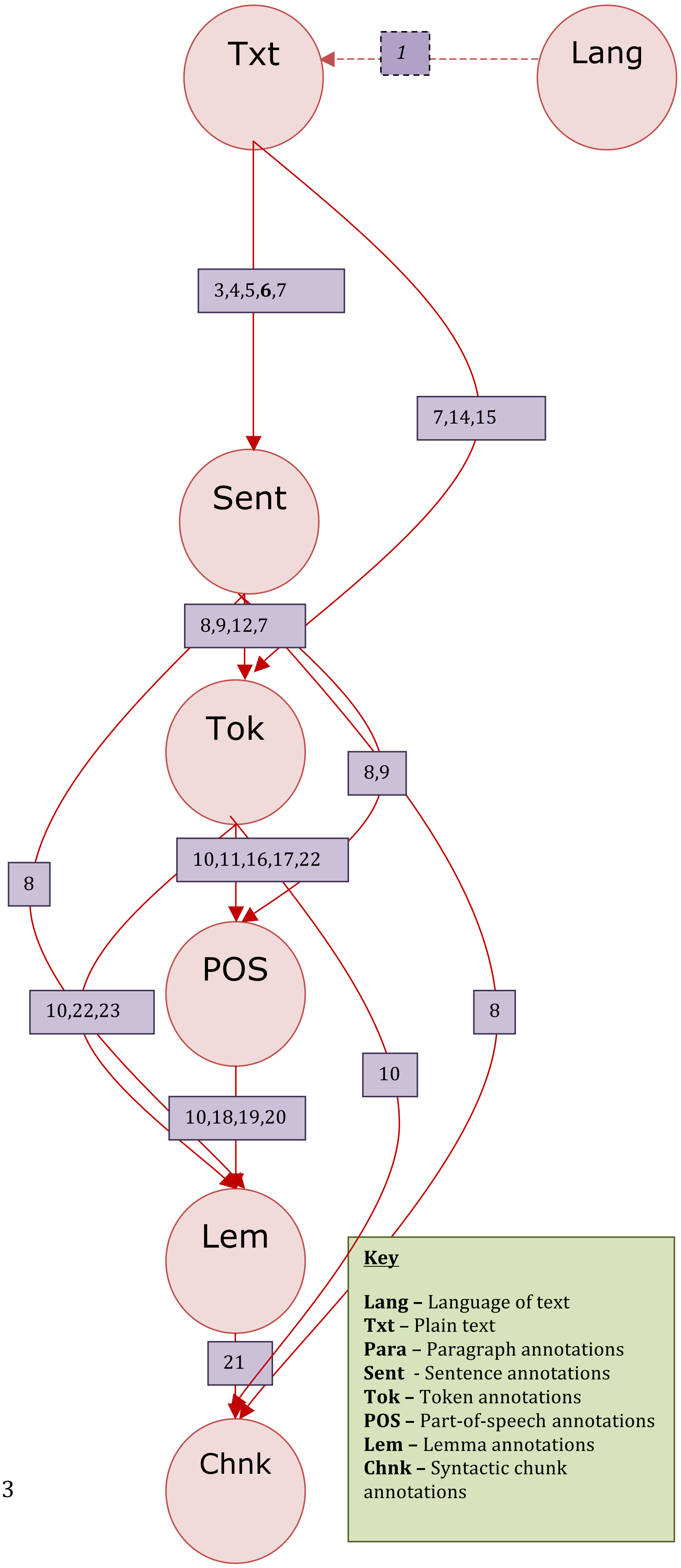
1 - RACAI:Lang Identifier
3 - UOM:MLRS Sentence Splitter:Any
4 -UNIMAN:Genia Sentence Splitter: en
5- UNIMAN:OpenNLP sentence detector: en
6- **UNIMAN: Cafetiere sentence splitter: en**
7- RACAI:TTL-Tokenizer:ro,en,fr
8 - UNIMAN:Genia Tagger (with tokenization): en
9 - UNIMAN:Stepp Tagger (with tokenization): en
10 - UNIMAN:Genia Tagger (no tokenization): en
11 - UNIMAN:Stepp Tagger (no tokenization): en
12 - UNIMAN:OpenNLP tokenizer
14 - UAIC: TokenizerUAIC: Any
15 - UNIMAN: Apertium Morpho Analyser: en,es,ca,pt,gl,eu
16 - UNIMAN:OpenNLP Tagger:en
17 - RACAI:TTL Tagger:ro,en,fr
18 - RACAI: TTL Lemmatizer: ro,en,fr
19- UAIC: Lemmatizer-UAIC: ro
20 - UNIMAN:morpha:en
21 - UPF: freeling_morpho: es,ca
22 -UNIMAN: Apertium Tagger: en,es,ca,pt,gl,en
24- UAIC: Lemmatizer-UAIC_v1: ro

Key

Lang – Language of text
Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations
POS – Part-of-speech annotations
Lem – Lemma annotations

Syntactic chunking

Purpose: Identifies syntactic chunks in plain text
Languages: En, Ro, Fr.



Resources

1 - RACAI:Lang Identifier
3 - UOM:MLRS Sentence Splitter:Any
4 - UNIMAN:Genia Sentence Splitter: en
5- UNIMAN:OpenNLP sentence detector: en
6 - **UNIMAN: Cafetiere sentence splitter: en**
7- RACAI:TTL-Tokenizer:ro,en,fr
8 - UNIMAN:Genia Tagger (with tokenization): en
9 - UNIMAN:Stepp Tagger (with tokenization): en
10 - UNIMAN:Genia Tagger (no tokenization): en
11 - UNIMAN:Stepp Tagger (no tokenization): en
12 - UNIMAN:OpenNLP tokenizer:en
14 - UAIC: TokenizerUAIC: Any
15 - UNIMAN: Apertium Morpho Analyser: en
16 - UNIMAN:OpenNLP Tagger:en
17 - RACAI:TTL Tagger:ro,en,fr
18 - RACAI: TTL Lemmatizer: ro,en,fr
19 - UAIC: Lemmatizer-UAIC: ro
20 - UNIMAN:morpha:en
21 - RACAI: TTL Chunker: ro,en,fr
22 - UNIMAN: Apertium Tagger: en

Key

Lang – Language of text
Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations
POS – Part-of-speech annotations
Lem – Lemma annotations
Chnk – Syntactic chunk annotations

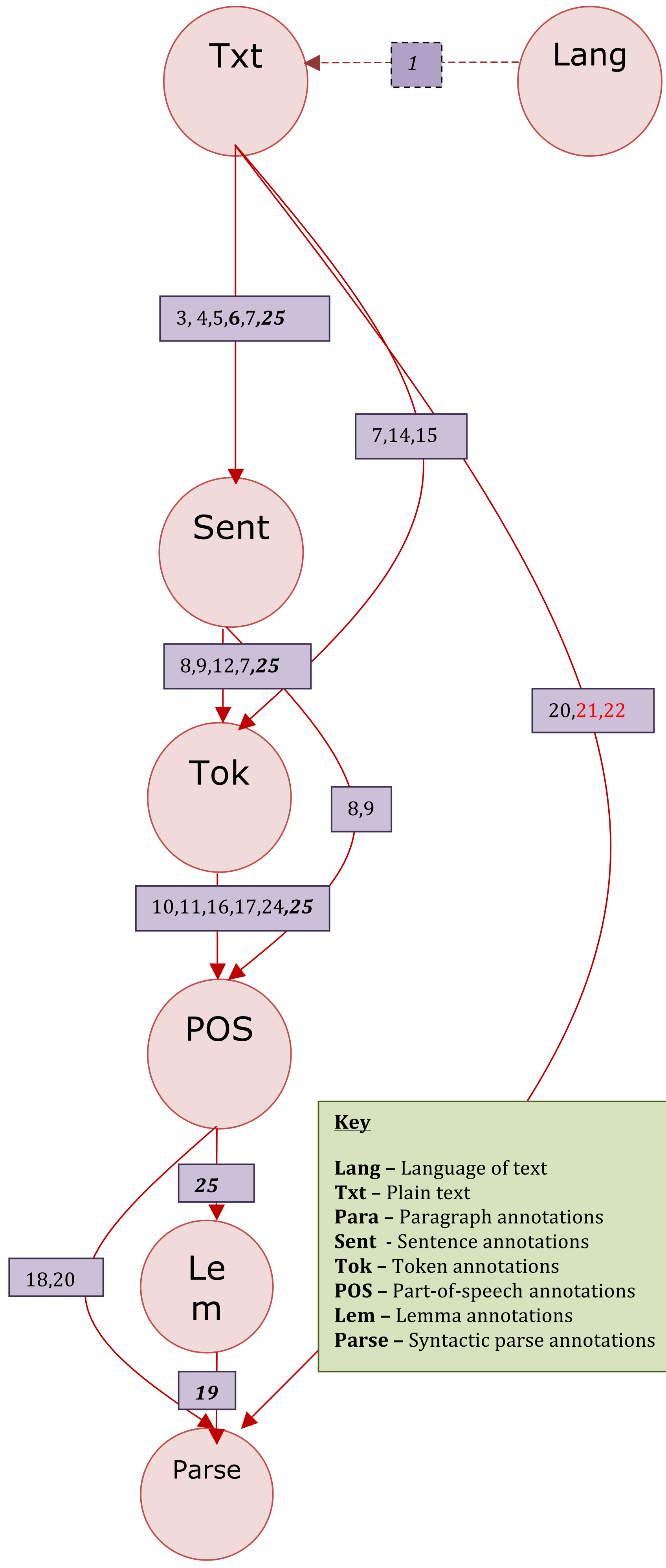
Syntactic parsing

Purpose: Performs syntactic parsing on plain text

Languages: En, **Es, Ca,** *Ro*

Resources

1 - RACAI:Lang Identifier
3 - UOM:MLRS Sentence Splitter:Any
4 -UNIMAN:Genia Sentence Splitter: en
5- UNIMAN:OpenNLP sentence detector: en
6 - **UNIMAN: Cafetiere sentence splitter**
7- RACAI:TTL Tokenizer:ro,en,fr
8 - UNIMAN:Genia Tagger (with tokenization): en
9 - UNIMAN:Stepp Tagger (with tokenization): en
10 - UNIMAN:Genia Tagger (no tokenization): en
11 - UNIMAN:Stepp Tagger (no tokenization): en
12 - UNIMAN:OpenNLP tokenizer:en
14 - UAIC: TokenizerUAIC: Any
15 - UNIMAN: Apertium Morpho Analyser: en
16 - UNIMAN:OpenNLP Tagger:en
17 - RACAI:TTL Tagger:ro,en,fr
18- UNIMAN: Enju Parser (HPSG): en
19 - UAIC: FDG-Parser-UAIC:ro
20 - UNIMAN: Stanford Parser:en
21 - UPF: freeling_parsed: es,ca
22 - UPF: freeling_dependency: es,ca
24 - UNIMAN: Apertium Tagger: en
25 – UAIC: PosTagger-UAIC:ro

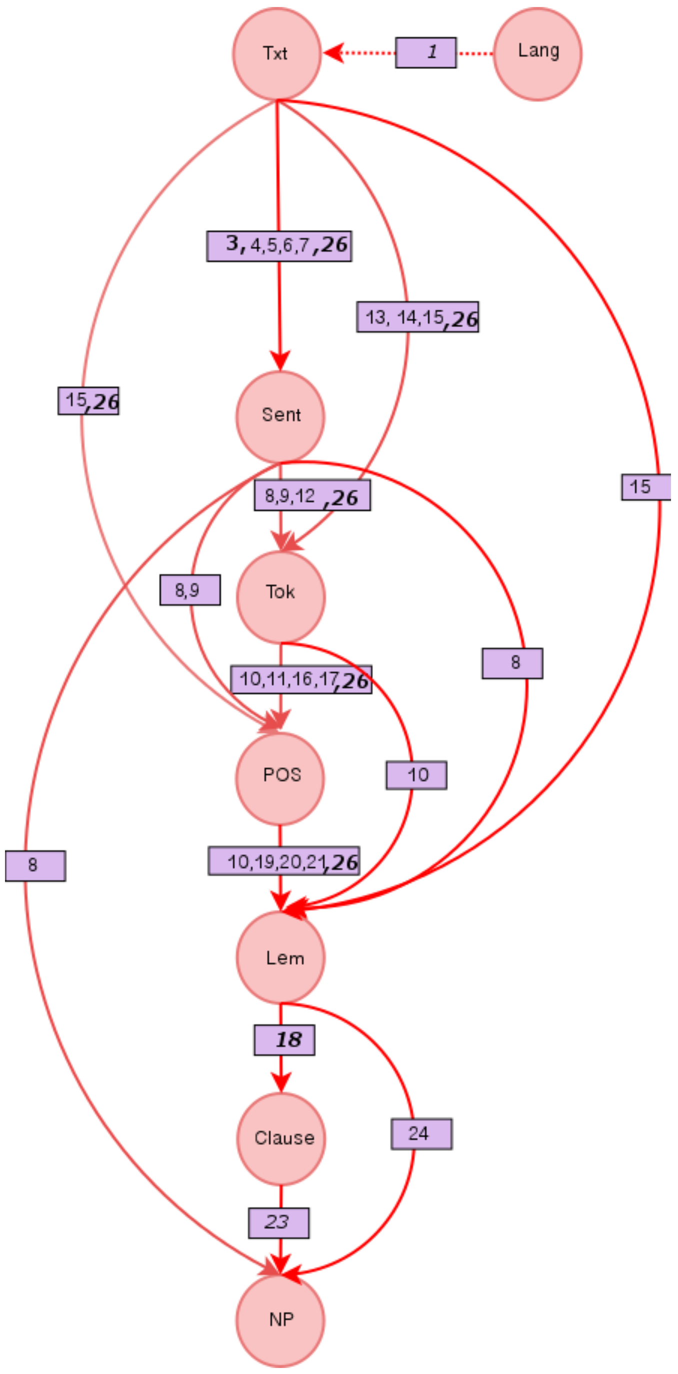


Key

Lang – Language of text
Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations
POS – Part-of-speech annotations
Lem – Lemma annotations
Parse – Syntactic parse annotations

NP chunking

Purpose: Identifies noun phrase chunks in plain text
Languages: En Ro, Fr



Resources

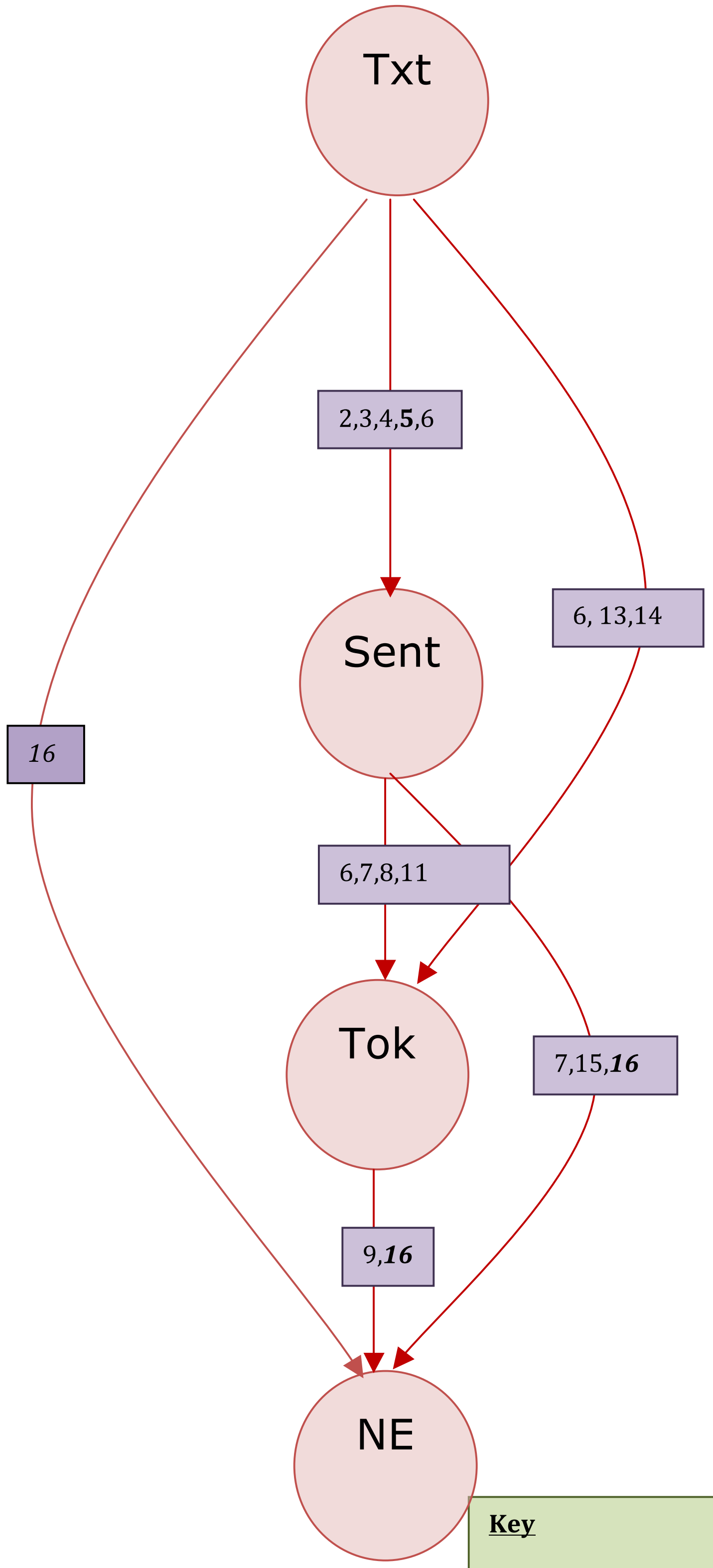
1 - RACAI:Lang Identifier
3 - UOM:MLRS Sentence Splitter:Any
4 - UNIMAN:Genia Sentence Splitter: en
5- UNIMAN:OpenNLP sentence detector: en
6 - UNIMAN: Cafetiere sentence splitter
7- RACAI:TTL Tokenizer:ro,en,fr
8 - UNIMAN:Genia Tagger (with tokenization): en
9 - UNIMAN:Stepp Tagger (with tokenization): en
10 - UNIMAN:Genia Tagger (no tokenization): en
11 - UNIMAN:Stepp Tagger (no tokenization): en
12 - UNIMAN:OpenNLP tokenizer:en
13 - RACAI:TTL Tokenizer:ro,en
14 - UAIC: TokenizerUAIC: **Any**
15 - UNIMAN: Apertium Morpho Analyser: en,ro
16 - UNIMAN:OpenNLP Tagger:en
17 - RACAI:TTL Tagger:ro,en,fr
18 -UAIC: Splitter-UAIC_v2:ro
19 - RACAI: TTL Lemmatizer: ro,en,fr
20 - UAIC: Lemmatizer-UAIC: ro
21 - UNIMAN:morpha:en
23 – UAIC:NP-Chunker-UAIC:ro
24 – RACAI:TTL-Chunker:ro,en
25 - UNIMAN: Apertium Tagger: en,ro,fr
26 – UAIC: PosTagger-UAIC:ro

Key

Lang – Language of text
Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations
POS – Part-of-speech annotations
Lem – Lemma annotations
Clause – Clause annotations
FDG- FDG parse annotations
NP – Noun phrase annotations

Named entity recognition

Purpose: Identifies named entities within plain text. UNIMAN's GENIA tagger and NEMine recognise biomedical named entities. IST's Named Entity recognizer is trainable for different languages and entity types.
Languages: En (biomedical), *Others (using IST's NR recognizer)*



Resources

2 - UOM:MLRS Sentence Splitter:Any
3 -UNIMAN:Genia Sentence Splitter: en
4- UNIMAN:OpenNLP sentence detector: en
5 - UNIMAN:Cafetiere Sentence Splitter:en
6 - RACAI:TTL-Tokenizer:en
7 - UNIMAN:Genia Tagger (with tokenization): en
8 - UNIMAN:Stepp Tagger (with tokenization): en
9 - UNIMAN:Genia Tagger (no tokenization): en
10 - UNIMAN:Stepp Tagger (no tokenization): en
11 - UNIMAN:OpenNLP tokenizer:en
12 - RACAI:TTL Tokenizer:en
13 - UAIC: TokenizerUAIC: Any
14 - UNIMAN: Apertium Morpho Analyser: en
15 - UNIMAN:NEMine:en
16 - IST:Named Entity Recognizer: trainable for different languages

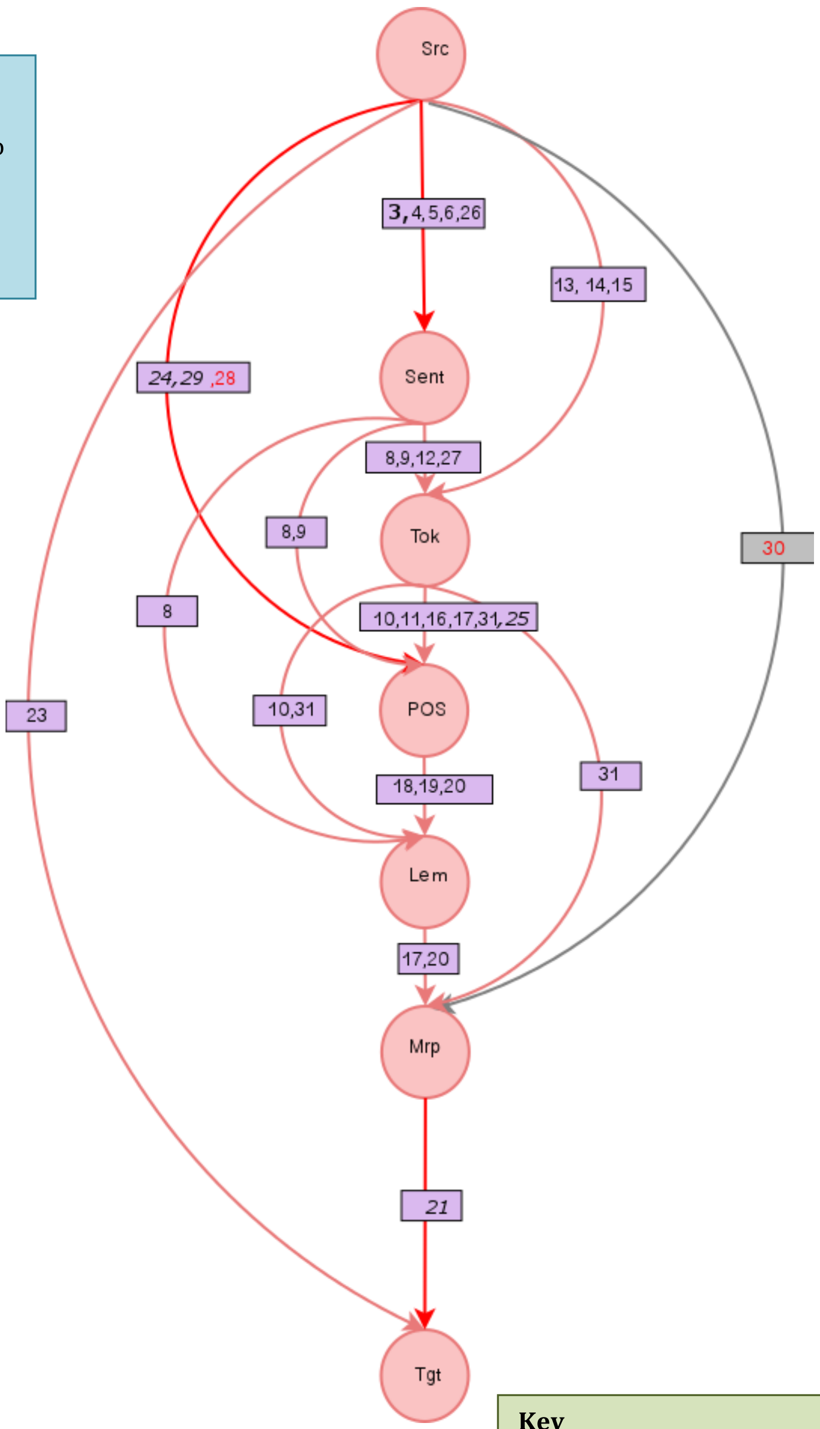
Key

Lang – Language of text
Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations
NE – Named Entity annotations

Text translation

Purpose: Translates text from one language to another

Languages Pairs: *En<->Es, Ca <->Es, Es<->Gl, Es<->Pt, Eu->Es, En<->Gl,*



Resources

3 - UOM:MLRS Sentence Splitter:Any
4 - UNIMAN:Genia Sentence Splitter: en
5- UNIMAN:OpenNLP sentence detector: en
6 - UNIMAN:NaCTeM sentence breaker:en
8 - UNIMAN:Genia Tagger (with tokenization): en
9 - UNIMAN:Stepp Tagger (with tokenization): en
10 - UNIMAN:Genia Tagger (no tokenization): en
11 - UNIMAN:Stepp Tagger (no tokenization): en
12 - UNIMAN:OpenNLP tokenizer:en
13 - RACAI:TTL Tokenizer:ro,en
14 - UAIC: TokenizerUAIC: Any
15 - UNIMAN: Apertium Morpho Analyser: en,es,pt,gl,eu,
16 - UNIMAN:OpenNLP Tagger:en
17 - RACAI:TTL Tagger:ro,en,fr
18 - RACAI: TTL Lemmatizer: ro,en,fr
19- UAIC: Lemmatizer-UAIC: ro
20 - UNIMAN:morpha:en
21 - UNIMAN:Apertium Translator:All language pairs/directions shown above
23 - UPC: N-II: En<->Es, Es<->Ca
24 - ULX:LXTagger with tokenization:pt
25 - ULX:Pos Tagger:pt
26 - ULX:Chunker:pt
27 - ULX:Tokenizer:pt
28 - UPF:freeling_tagging:es,ca
29 - UPF:iula_tagger:es,ca
30- UPF: freeling_morpho: es,ca
31 - UNIMAN: Apertium Tagger: en,es,pt,gl,eu

Key

Lang – Language of text
Src – Source language text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations
POS – Part of speech annotations
Lem – Lemma annotations
Mrp – Morphological annotations
Tran – Translated morphological structures
Tgt – Target language text

5.2 Newly implemented workflows

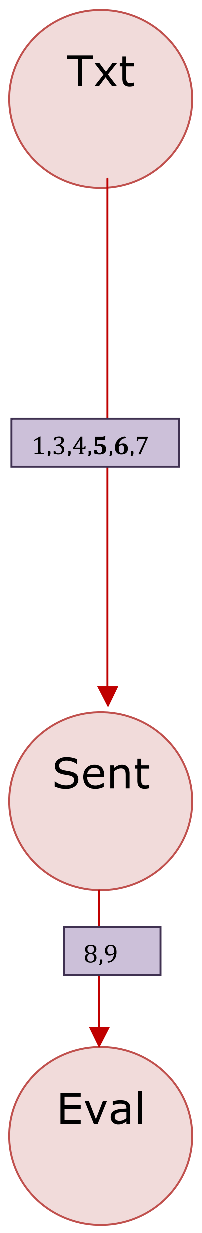
In this section, we provide the diagrams corresponding to the 10 new workflows that can now be implemented as a result of the UIMA components available as part of the current deliverable. These include 4 evaluation workflows that make use of the new corpus reader components. As in the previous section, certain features of the diagrams indicate the current progress of the work, in terms of the number of paths that can currently be taken through the workflows, and which languages are supported.

The diagrams are based on those originally provided in D2.2. The features of these diagrams are as follows:

- Resources that have already been wrapped are shown in black text.
- Resources that have not yet been wrapped as UIMA components are shown using red text
- To show which of the planned transitions between information states that are not currently possible at all (due to a lack of available wrapped components), grey arrows are shown between the information states.
- Languages that were planned for a particular workflow, but which are not currently possible are shown in red.
- Any changes to planned components or their usage since the D2.2 are indicated using bold font. This may include extra components that were not originally planned, or else a change in the planned usage of the component (i.e., it appears at a different place in the workflow).

Sentence splitting evaluation

Purpose: Evaluates sentence splitting performance against plain text
Languages: En, Pt



Resources

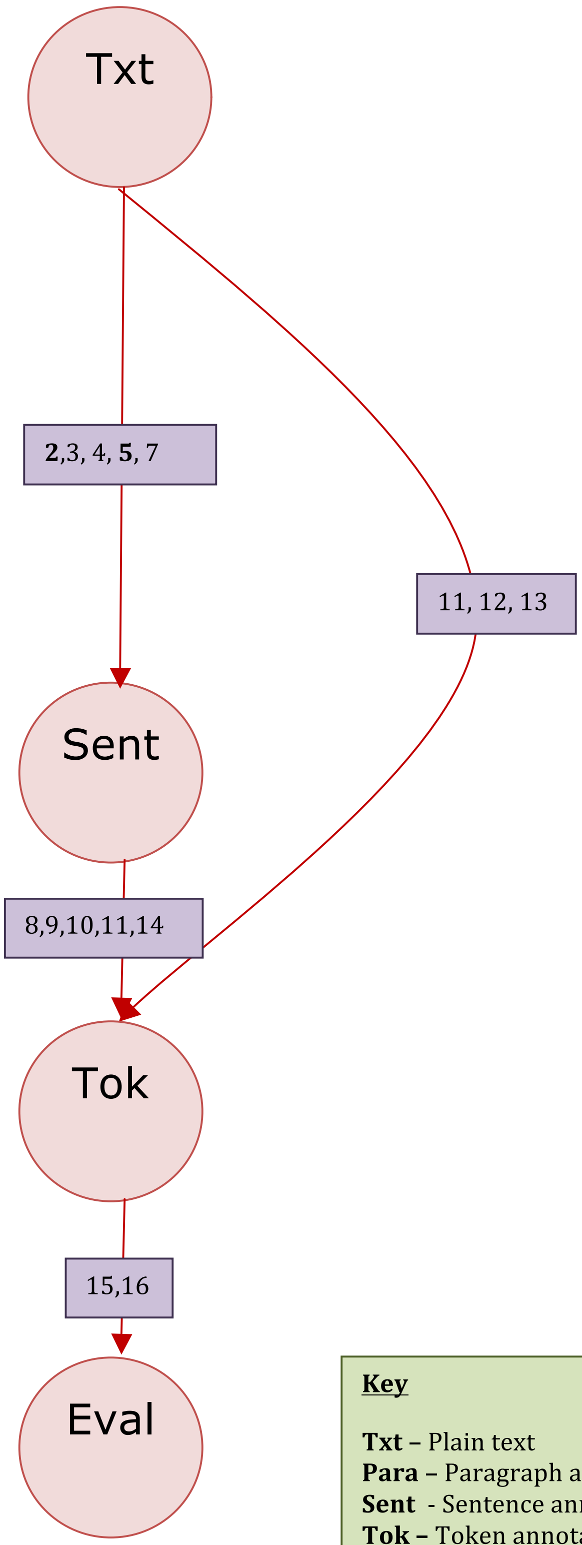
1 - ULX: Chunker: pt
3 - UNIMAN: GENIA sentence splitter:en
4 - UNIMAN: OpenNLP sentence detector: en
5 - UNIMAN: Cafetiere Sentence Splitter: en
6 - UOM: Sentence Splitter: Any
7 - RACAI: Sentence splitter: en
8 - UNIMAN: GENIA corpus:en
9 - ULX: CINTIL corpus:pt

Key

Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Eval – Evaluation results

Tokenization evaluation

Purpose: Evaluates tokenization performance against gold standard corpus
Languages: En, Pt



Resources

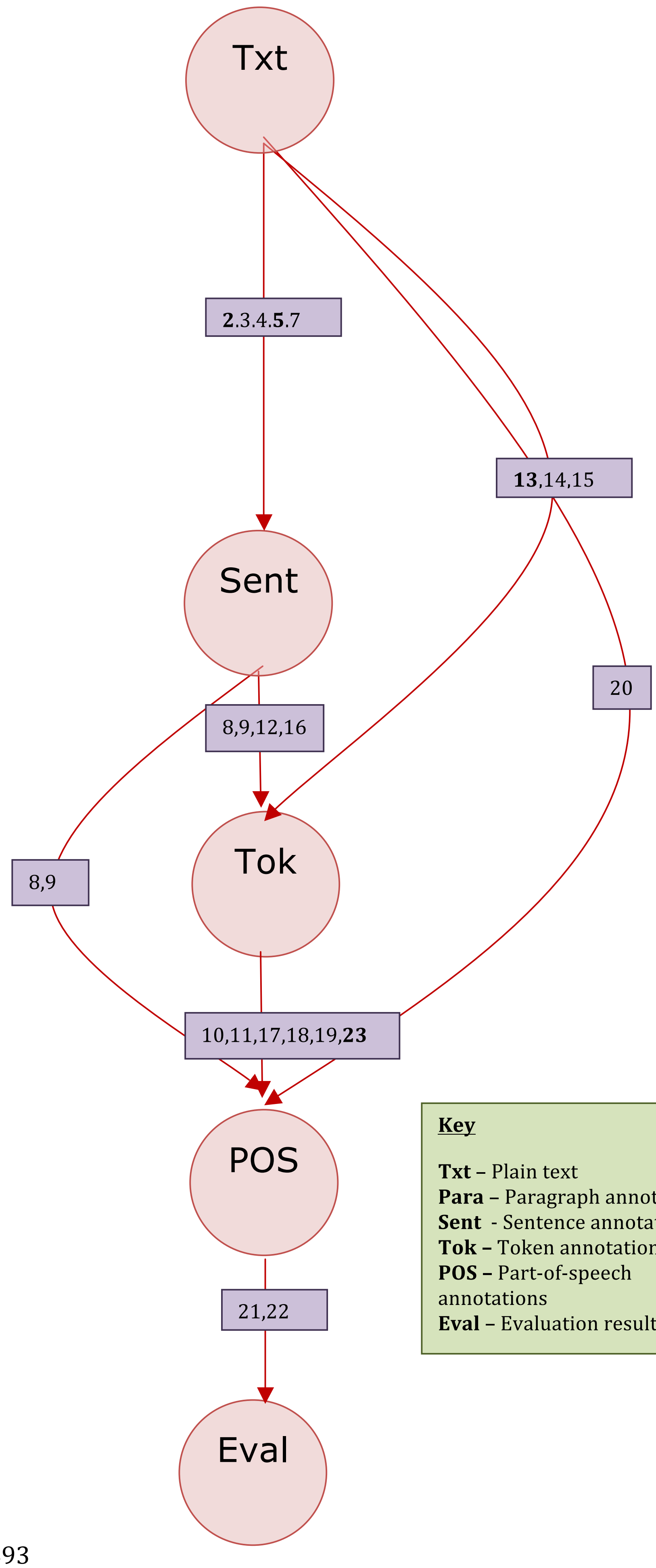
2 - UOM:Sentence Splitter:Any
3 - UNIMAN:Genia Sentence Splitter: en
4 - UNIMAN:OpenNLP sentence detector: en
5 - UNIMAN: Cafetiere Sentence Splitter: en
7 - ULX: Chunker: pt
8 - UNIMAN:Genia Tagger (with tokenization): en
9 - UNIMAN:Stepp Tagger (with tokenization): en
10 - UNIMAN:OpenNLP tokenizer:en
11 - RACAI:TTL Tokenizer:en
12 - UAIC: TokenizerUAIC: en
13- UNIMAN: Apertium Morpho Analyser: en,pt
14 - ULX: Tokenizer:pt
15 - UNIMAN: GENIA corpus:en
16- ULX: CINTIL corpus:pt

Key

Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations
Eval – Evaluation results

Part-of-speech tagging evaluation

Purpose: Evaluates part-of-speech tagging performance against gold standard corpus
Languages: En, Pt.



Resources

2 - UOM:Sentence Splitter:Any
3 - UNIMAN:Genia Sentence Splitter: en
4 - UNIMAN:OpenNLP sentence detector: en
5 - UNIMAN:Cafetiere Setence Splitter:en
7 - ULX:Chunker:pt
8 - UNIMAN:Genia Tagger (with tokenization): en
9 - UNIMAN:Stepp Tagger (with tokenization): en
10 - UNIMAN:Genia Tagger (no tokenization): en
11 - UNIMAN:Stepp Tagger (no tokenization): en
12 - UNIMAN:OpenNLP tokenizer:en
13 - RACAI:TTL Tokenizer:en
14 - UAIC: TokenizerUAIC: en
15 - UNIMAN: Apertium Morpho Analyser: en,pt
16 -ULX: Tokenizer:pt
17 - UNIMAN:OpenNLP Tagger:en
18 - RACAI:TTL Tagger:en
19 - ULX: LX-POSTagger:pt
20 – ULX:LX-Tagger:pt
21 - UNIMAN: GENIA corpus:en
22- ULX: CINTIL corpus:pt
23 - UNIMAN: Apertium Tagger: en,pt

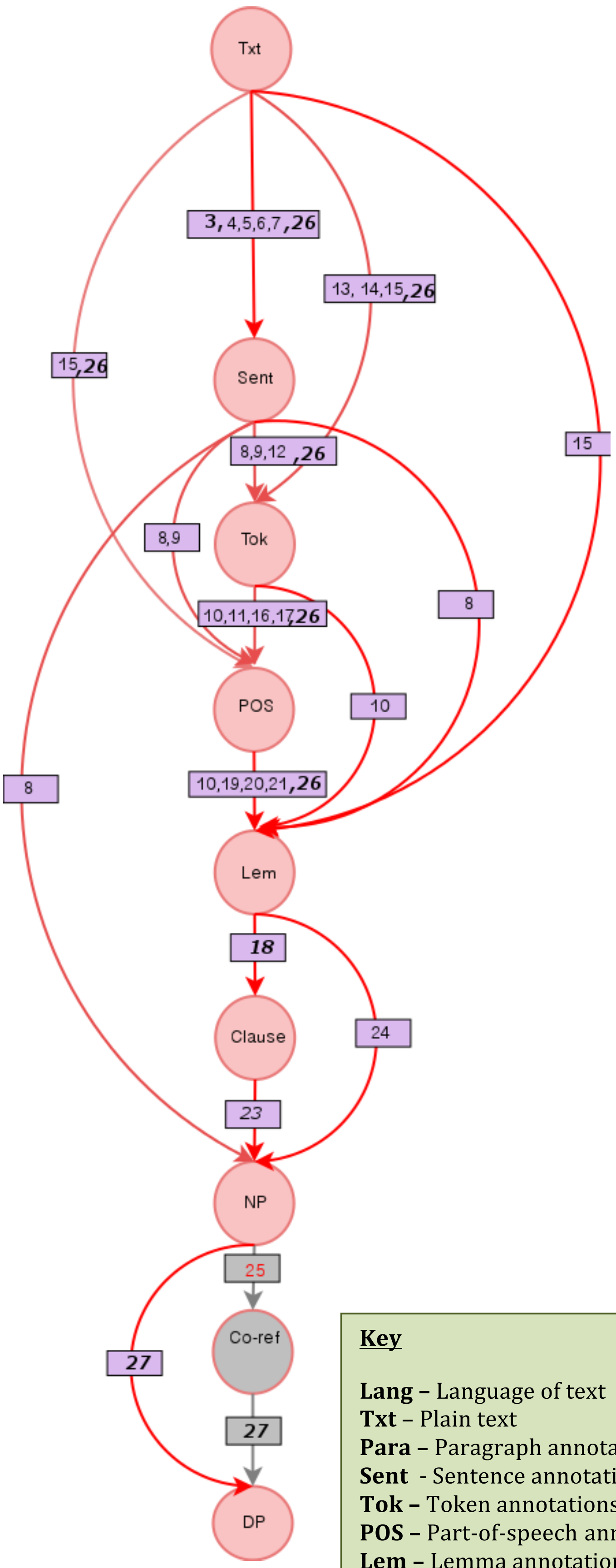
Key

Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations
POS – Part-of-speech annotations
Eval – Evaluation results

Discourse parsing

Purpose: Performs discourse parsing on plain text

Languages: En, Ro.



Resources

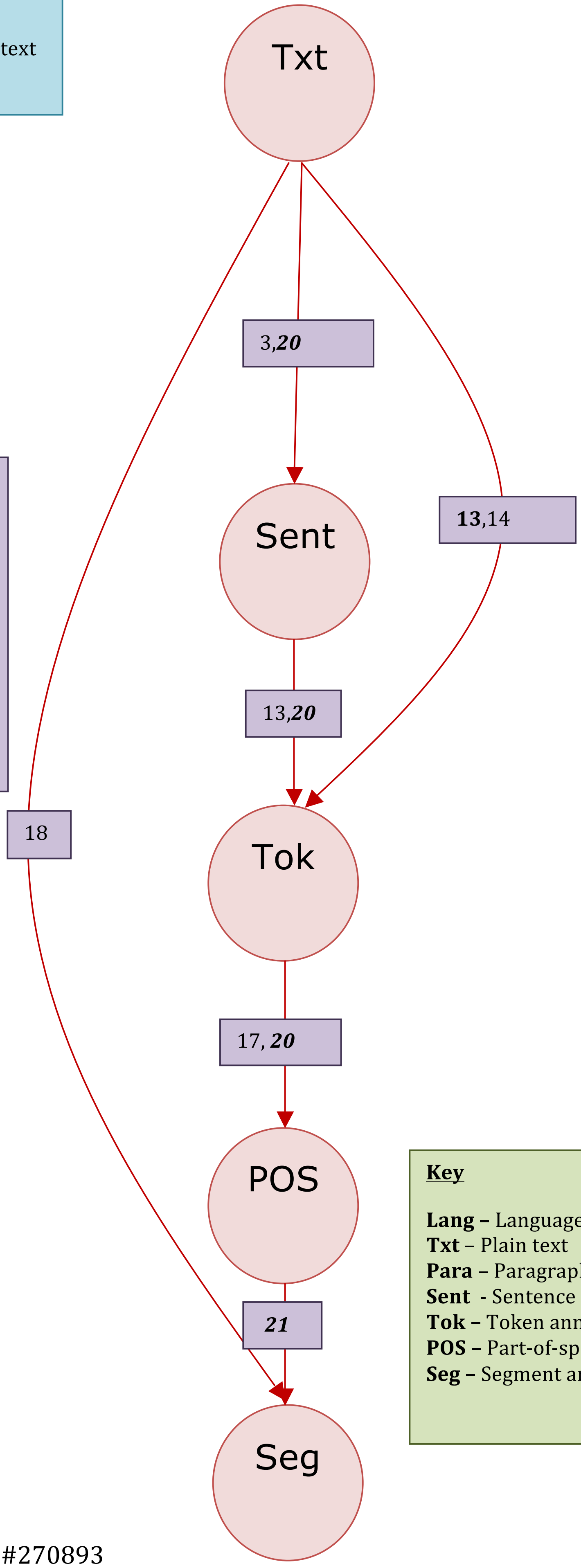
1 - RACAI:Lang Identifier
2 - UOM:Paragraph Breaker:Any
3 - UOM:Sentence Splitter:Any
4 -UNIMAN:Genia Sentence Splitter: en
5- UNIMAN:OpenNLP sentence detector: en
6 - UNIMAN:NaCTeM sentence breaker:en
7- RACAI: Sentence Splitter:ro,en
8 - UNIMAN:Genia Tagger (with tokenization): en
9 - UNIMAN:Stepp Tagger (with tokenization): en
10 - UNIMAN:Genia Tagger (no tokenization): en
11 - UNIMAN:Stepp Tagger (no tokenization): en
12 - UNIMAN:OpenNLP tokenizer:en
13 - RACAI:TTL Tokenizer:ro,en
14 - UAIC: TokenizerUAIC: ro, en
15 - UNIMAN: Apertium Morpho Analyser: en
16 - UNIMAN:OpenNLP Tagger:en
17 - RACAI:TTL Tagger:ro,en,fr
18 -UAIC: Splitter-UAIC_v1:ro
19 - RACAI: TTL Lemmatizer: ro,en,
20- UAIC: Lemmatizer-UAIC: ro
21 - UNIMAN:morpha:en
22 - UAIC: Splitter-UAIC:ro,en
23 – UAIC:NP-Chunker-UAIC:ro
24 – RACAI:TTL-Chunker:ro,en
25 – UAIC: RARE-UAIC:ro,en
26 – UAIC: PosTagger-UAIC:ro
27 – UAIC: DP-UAIC:ro,en

Key

Lang – Language of text
Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations
POS – Part-of-speech annotations
Lem – Lemma annotations
Seg – Segment annotations
FDG- FDG parse annotations
NP – Noun phrase annotations
Co-ref – Co-reference annotations
DP – Discourse Parse annotations

Segmentation

Purpose: Identifies segments within plain text
Languages: En, Ro



Resources

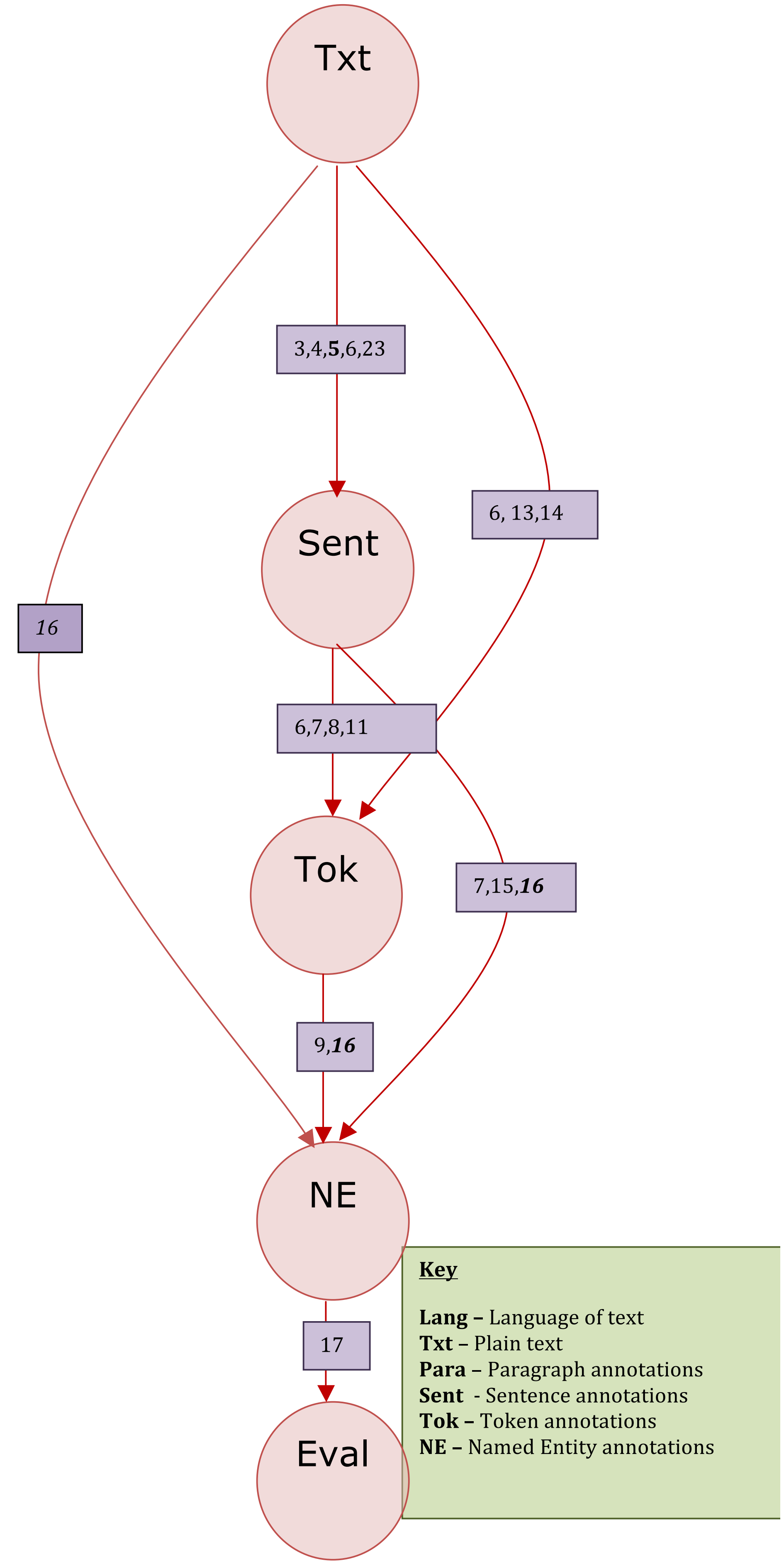
3 - UOM:Sentence Splitter:Any
13 - RACAI:TTL Tokenizer:ro,en
14 - UAIC: TokenizerUAIC: ro, en
17 - RACAI:TTL Tagger:ro,en,fr
18 - UAIC: Splitter-UAIC_v1:en,ro
20 – UAIC: PosTagger-UAIC:ro
21 - UAIC: Splitter-UAIC_v2:ro

Key

Lang – Language of text
Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations
POS – Part-of-speech annotations
Seg – Segment annotations

Named entity recognition evaluation

Purpose: Identifies named entities within plain text. UNIMAN's GENIA tagger and NEMine recognise biomedical named entities. IST's Named Entity recognizer is trainable for different languages and entity types.
Languages: En (biomedical), *Others (using IST's NR recognizer)*



Resources

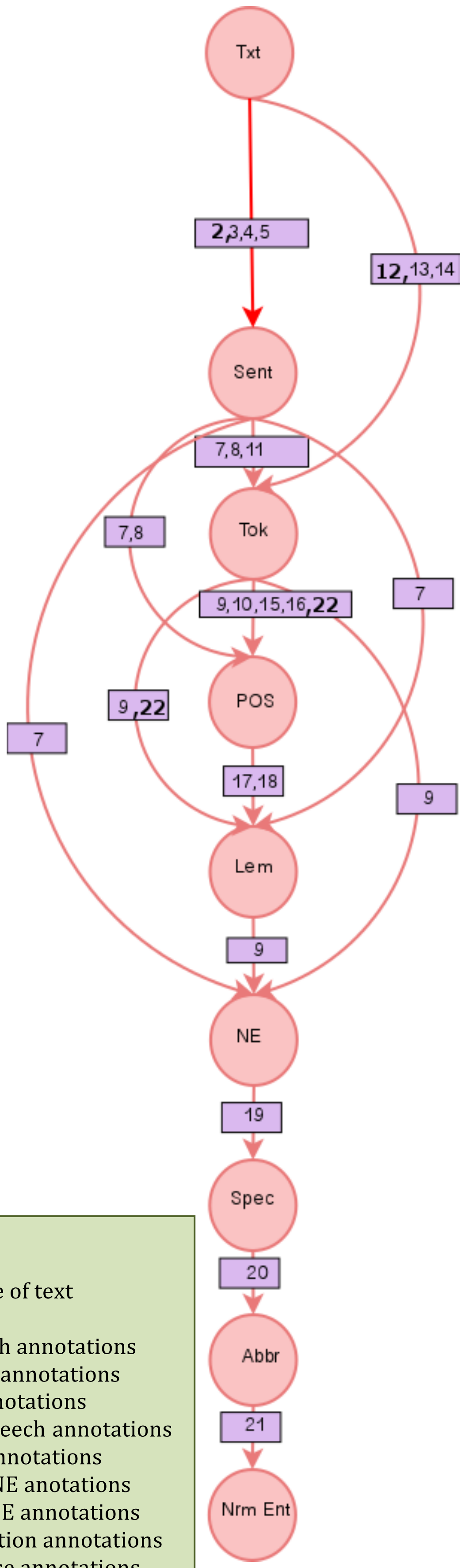
1 - UOM:MLRS Paragraph Breaker:Any
2 - UOM:MLRS Sentence Splitter:Any
3 -UNIMAN:Genia Sentence Splitter: en
4- UNIMAN:OpenNLP sentence detector: en
5 - UNIMAN:Cafetiere Sentence Splitter:en
6 - RACAI:TTL-Tokenizer:en
7 - UNIMAN:Genia Tagger (with tokenization): en
8 - UNIMAN:Stepp Tagger (with tokenization): en
9 - UNIMAN:Genia Tagger (no tokenization): en
10 - UNIMAN:Stepp Tagger (no tokenization): en
11 - UNIMAN:OpenNLP tokenizer:en
12 - RACAI:TTL Tokenizer:en
13 - UAIC: TokenizerUAIC: Any
14 - UNIMAN: Apertium Morpho Analyser: en
15 - UNIMAN:NEMine:en
16 - IST:Named Entity Recognizer: trainable for different languages
17 – UNIMAN:GENIA corpus
23 – UOM:Sentence Splitter:Any

Key

Lang – Language of text
Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations
NE – Named Entity annotations

Species disambiguation for biological named entities

Purpose: Identifies biological named entities and disambiguates them according to species, by assigning a species ID from the NCBI taxonomy
Languages: En



Resources

3 -UNIMAN:Genia Sentence Splitter: en
4- UNIMAN:OpenNLP sentence detector: en
5 - UNIMAN:NaCTeM sentence breaker:en
7 - UNIMAN:Genia Tagger (with tokenization): en
8 - UNIMAN:Stepp Tagger (with tokenization): en
9 - UNIMAN:Genia Tagger (no tokenization): en
10 - UNIMAN:Stepp Tagger (no tokenization): en
11 - UNIMAN:OpenNLP tokenizer:en
12 - RACAI:TTL Tokenizer:en
13 - UAIC: TokenizerUAIC:en
14 - UNIMAN: Apertium Morpho Analyser: en
15 - UNIMAN:OpenNLP Tagger:en
16 - RACAI:TTL Tagger:en
17 - RACAI: TTL Lemmatizer: en
18 - UNIMAN:morpha:en
19 - UNIMAN: NaCTeM species word detector:en
20 - UNIMAN: ExractAbbrev:en
21 - UNIMAN: NaCTeM Species Disambiguator:en
22 - UNIMAN: Apertium Tagger: en

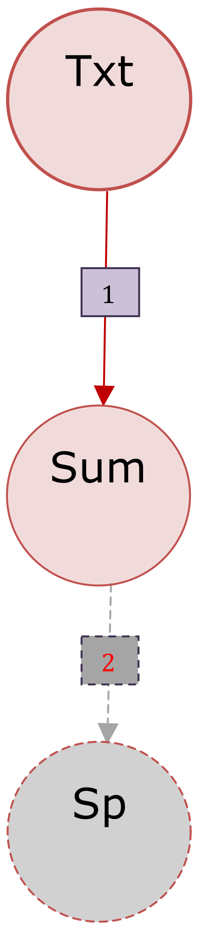
Key

Lang – Language of text
Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations
POS – Part-of-speech annotations
Lem – Lemma annotations
NE – Biological NE anotations
Spec – Species NE annotations
Abbr - Abbreviation annotations
NP – Noun phrase annotations
Co-ref – Co-reference annotations
Nrm Ent – Normalised species entity annotations, with NCBI taxonomy Ids attached

Summarization version 1 (with possible spoken output)

Purpose: Produces a summary of given plain text (and possibly reads it out)

Languages: Ro, En (only En for spoken output)



Resources

1 - UAIC: Summarizer-UAIC:ro,en
2 - UPC:Ogmios:en

Key

Txt – Plain text
Sum – Summary of text
Sp – Synthesized speech

6 Upcoming work

The work to be undertaken on interoperability in the final 6 months of the project will take two directions:

- Component/workflow implementation. In our original project plan, it was hoped that at this point in the project, all of the originally planned 67 workflows would be available as UIMA components allowing all 26 workflows to be built. However, at M18, only 50 components are available as UIMA components. There are a number of reasons why the remaining components are not yet available, which are detailed below. In some cases, experience gained with U-Compare has revealed that a few of the originally planned components are not particularly suitable for inclusion in workflows. However, there are plans to make the majority of the remaining components available during the remaining months of the project
- Integration of components. As has been mentioned above, this work will include the integration of the majority of the components (where licences allow) into the core library of U-Compare. The work will also include some upgrades to U-Compare itself, which are either required or desirable in order to run the workflows that can be built using the integrated components.

In the subsections below, we provide further details of these two directions of work

6.1 Implementation of remaining workflows

In this section, we list the resources that have not yet been implemented as UIMA components. For each resource, we provide reasons why the implementation has not yet taken place, and state whether or not there it is still planned to implement the resource as a UIMA component during the remaining months of the project. These remaining resources belong to 4 different partners, i.e., UAIC, UOM, UPC and UPF.

6.1.1 UAIC

In D2.2, UAIC identified 19 resources that were considered suitable to make available as UIMA components. They made available 3 UIMA components as part of D4.4 (including an extra resource, a second version of their lemmatizer), and have made a further 8 UIMA components available in the current deliverable (including 2 resources that were not originally envisaged in D2.2). According to an understanding agreed upon by the consortium when the work on UIMA/U-Compare began, only active resources (i.e. tools) that can be included within a workflow, or passive resources (e.g., corpora) that can be used to evaluate the output of workflows will be provided as U-Compare components during the lifetime of the project. The few UAIC resources that will not be integrated do not meet with this criterion, as detailed below:

Diacritics-UAIC

Description: Corrects Romanian diacritics in texts.
Languages covered: Romanian
Reason not implemented: Still undergoing testing
Planned implementation during final phase of project: Yes

SRL-UAIC

Description: Semantic role labeller
Languages covered: Romanian
Reason not implemented: Still undergoing testing
Planned implementation during final phase of project: Yes

RARE-UAIC

Description: RARE (Robust Anaphora Resolution Engine) is a system capable of building coreference chains using morphological and syntactic cues.
Languages covered: Romanian, English
Reason not implemented: Still undergoing testing
Planned implementation during final phase of project: Yes

TE-UAIC

Description: Textual entailment tool. The input is a pair of texts, one a hypothesis and the other a support text. The system checks whether the hypothesis is supported (can be extracted/inferred) by the support text. The output is yes/no.
Languages covered: Romanian, English
Reason not implemented: Still undergoing testing
Planned implementation during final phase of project: Yes

OccurrenceFinder-UAIC

Description: Finds occurrences of a word or annotation in context
Languages covered: Romanian, English
Reason not implemented: Not particularly suitable for use in text mining workflows
Planned implementation during final phase of project: No

OntologyBuilder-UAIC

Description: Builds basic ontologies from annotated data using patterns to identify keywords, definitions and semantic relations
Languages covered: Romanian, English
Reason not implemented: Does not fit the pattern of other implemented tools, which add annotations to input documents; U-Compare does not currently support the use of components that produce new resources
Planned implementation during final phase of project: No

QA-UAIC

Description: Question-Answering component
Languages covered: Romanian, English
Reason not implemented: Does not fit the pattern of other implemented tools, which add annotations to input documents
Planned implementation during final phase of project: No

QACorpus-UAIC

Description: Corpus of questions and answers based on the QA task at CLEF 2011. Text passages are provided with multiple choice answers, of which one is the correct answer. The correct answer is shown as an attribute.
Languages covered: Romanian
Reason not implemented: Annotated corpora are only useful to implement if they can act as gold standard data for annotations output by particular workflows. Since QA workflows are not currently supported, a corpus reader will not be made available for this corpus.
Planned implementation during final phase of project: No

RO-FDGBank

Description: Syntactically annotated corpus with
Languages covered: Romanian
Reason not implemented: Lack of time

Planned implementation during final phase of project: Yes

<u>RO-FN</u>

Description: FrameNet type annotation for Romanian sentences
Languages covered: Romanian
Reason not implemented: Lack of time
Planned implementation during final phase of project: Yes

<u>ROSemClass</u>

Description: Corpus in which words indicating sentiment (subjectivity) have been annotated.
Languages covered: Romanian
Reason not implemented: Currently no workflows possible for which this corpus could act as gold standard evaluation data
Planned implementation during final phase of project: No

6.1.2 UOM

UOM have only one planned component that is not yet available:

<u>POS-tagged corpus</u>

Description: Corpus with part-of-speech annotations
Languages covered: Maltese
Reason not implemented: Corpus is still under construction
Planned implementation during final phase of project: Yes

6.1.3 UPC

UPC also have one resource that has not yet been delivered as a UIMA component.

<u>Ogmios</u>

Description: Text-to-speech tool
Languages covered: Catalan, Spanish, English
Reason not implemented: Use of this tool in U-Compare is reliant on enhancements to U-Compare that will allow speech files to be played
Planned implementation during final phase of project: Yes

6.1.4 UPF

UPF are not directly involved in the UIMA/U-Compare work. Rather, they are involved with the PANACEA project, which is also concerned with interoperability, using web services. In order to foster collaboration between U-Compare and PANACEA, UNIMAN decided that it would endeavour to make certain PANACEA services available as U-Compare components, if their schedule allowed it (this work was not originally envisaged in the work plan). The web services selected deal with Spanish and Catalan, for which few other U-Compare components were planned to be made available. A total of 8 web services were identified, of which 2 have so far been implemented as UIMA components by UNIMAN. The time spent on these components so far has been largely been devoted to determining how to run these web services within the UIMA framework. This initial implementation work should make it more straightforward to implement the remaining 6 services as U-Compare components during the final phase of the project.

iula_preprocess

Description: Text preprocessing. Carries out: (i) text segmentation into minor structural units (titles, paragraphs, sentences, etc.); (ii) detection of entities not found in dictionaries (numbers, abbreviations, URLs, emails, proper nouns, etc.); and (iii) the keeping of sequences of two or more words in a single block (dates, phrases, proper nouns, etc.).
Languages covered: Spanish, Catalan
Reason not implemented: Time has been spent working out the framework for running these the UPF web services in UIMA/U-Compare; implementation should be straightforward.
Planned implementation during final phase of project: Yes

freeling_tokenizer

Description: Text tokenizer
Languages covered: Spanish, Catalan
Reason not implemented: Time has been spent working out the framework for running these the UPF web services in UIMA/U-Compare; implementation should be straightforward.
Planned implementation during final phase of project: Yes

freeling_morpho

Description: Morphological analyser
Languages covered: Spanish, Catalan

Reason not implemented: Time has been spent working out the framework for running these the UPF web services in UIMA/U-Compare; implementation should be straightforward.
Planned implementation during final phase of project: Yes

freeling_tagging

Description: Part-of-speech tagger
Languages covered: Spanish, Catalan
Reason not implemented: Time has been spent working out the framework for running these the UPF web services in UIMA/U-Compare; implementation should be straightforward.
Planned implementation during final phase of project: Yes

freeling_parsed

Description: Shallow parser
Languages covered: Spanish, Catalan
Reason not implemented: Time has been spent working out the framework for running these the UPF web services in UIMA/U-Compare; implementation should be straightforward.
Planned implementation during final phase of project: Yes

freeling_dependency

Description: Dependency parser
Languages covered: Spanish, Catalan
Reason not implemented: Time has been spent working out the framework for running these the UPF web services in UIMA/U-Compare; implementation should be straightforward.
Planned implementation during final phase of project: Yes

6.2 Integration of UIMA components into U-Compare

As mentioned above, there are two main parts to the integration work. Firstly, where their licences permit, most components will be integrated into U-Compare's core library. This will mean that the majority of the METANET4U resources that have been made available as UIMA components will be immediately available for use after U-Compare has been downloaded, so that workflows for a number of different languages can created with the minimum of effort.

The second part of the integration work will include enhancements to U-Compare itself. These enhancements will allow some new types of workflows that can be built using the new UIMA components to be more fully supported by U-Compare, in that there will be more complete facilities to visualise their outputs. These enhancements, which were first suggested in D4.4, can be summarised as follows:

- As has been mentioned above, two types of components that have been created during METANET4U make use of a new "view" of the text (i.e., a second sofa) to store their output. These are the components that carry out machine translation (in which the second sofa is used to store the target language text) and summarisation (in which the second sofa is used to store the summary of the text). U-Compare's annotation viewer does not currently support the visualisation of different sofas. Hence, UNIMAN plans to implement a new annotation viewer, which can display multiple views of a document side by side.
- U-Compare can currently only handle the construction of workflows that process one document at time and then move on to the processing of the next document. That is to say, the annotations in the UIMA CAS are cleared after the processing of each document is complete. Whilst this model of execution is suitable for many simpler types of processing, it cannot support certain more complex components and workflows. As an example, UAIC's automatic summarisation tool, which is currently wrapped a UIMA component that generates summaries of individual documents, can also generate a single summary of multiple documents. Within the UIMA framework, this latter type of behaviour would require information from CASes produced for individual documents to be merged together, in order to allow a summary for the complete document set to be created. The UIMA framework itself can support such behaviour, through the provision of provides CAS multipliers and CAS mergers, but these are not currently handled in U-Compare. UNIMAN will look into the feasibility of handling CAS multipliers/mergers within U-Compare, in order to allow more complex workflows to be created.
- In order to be able to experiment in U-Compare with workflows that produce speech based output, a new annotation "viewer" component will have to be implemented that is able to play speech-based annotations.

7 References

Ananiadou, S., Thompson, P., Kano, Y., McNaught, J., Attwood, T. K., Day, P. J. R., Keane, J., Jackson, D. and Pettifer, S.. (2011). Towards Interoperability of European Language Resources. Ariadne, 67.

Thompson, P., Kano, Y., McNaught, J., Pettifer, S., Attwood, T. K., Keane, J. and Ananiadou, S.. (2011). Promoting Interoperability of Resources in META-SHARE. In: Proceedings of the IJCNLP Workshop on Language Resources, Technology and Services in the Sharing Paradigm (LRTS), pp. 50-58.

Ferrucci, D., Lally, A., Gruhl, D., Epstein, E., Schor, M., Murdock, J. W. (2006). "Towards an Interoperability Standard for Text and Multi-Modal Analytics". *IBM Research Report RC24122*.

Kano, Y., Miwa, M., Cohen, K. B., Hunter, L. E., Ananiadou, S., & Tsujii, J. (2011). "U-Compare: A modular NLP workflow construction and evaluation system". *IBM Journal of Research and Development, 55*(3), 11:11-11:10.

Kano, Y., Baumgartner, W. A., Jr., McCrohon, L., Ananiadou, S., Cohen, K. B., Hunter, L. (2009). "U-Compare: share and compare text mining tools with UIMA". *Bioinformatics,* vol. 25, no. 15, 1997-1998.