First Version of Pilot Applications

Deliverable D4.4

Version 1.0

2012-02-01

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 to have 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 programmes; 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 to collaborate with other projects and where useful with other relevant multi-national forums or activities. This includes also to help build and operate broad inter-connected repositories and exchange facilities.
dissemination: to mobilise 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 an European project aiming at supporting language technology for European languages and multilingualism. It is a project in the META-NET Network of Excellence (www.meta-net.eu), 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.

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

METANET4U

First version of pilot applications

Document METANET4U-2012-D4.4
EC CIP project #270893

Deliverable D4.4
Completion: Final
Status: Submitted
Dissemination level: Restricted to other programme participants

Responsible: Sophia Ananiadou (WPS coordinator)

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

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

Reviewers: Mike Rosner, Andrew Attard

© all rights reserved by FCUL on behalf of METANTE4U

Contents

1 Introduction 6
2 Creation of UIMA components 8
3 U-Compare 9
3.1 Constructing workflows 9
3.2 Executing workflows and viewing results 11
3.3 U-Compare type system 12
4 Work on UIMA components and U-Compare workflows in METANET4U 13
5 Resources wrapped as UIMA components 15
5.1 University of Lisbon (ULX) 18
Tools 18
5.2 IST – Instituto Superior Técnico 20
5.3 University of Manchester – UNIMAN 20
Tools 20
5.4 University Alexandru Ioan Cuza (UAIC) 29
Tools 29
5.5 RACAI – Romanian Academy 30
Tools 30
5.6 University of Malta (UOM) 33
Tools 33
5.7 UPC - Universitat Politècnica de Catalunya 35
Tools 35
5.8 UPF- Universitat Pompeu Fabra 36
6 Workflows 36
Paragraph breaking 40
Sentence splitting 41
Tokenization 42
Part-of-speech tagging 43
Lemmatization 44
Syntactic chunking 45
Syntactic parsing 46
NP chunking 47
Named entity recognition 48
Text translation 49
7 Upcoming work 50

8 References 51

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.

Developers often combine together more basic LRs in order to develop more complex tools or applications for natural language processing (NLP). For example, the processes of sentence splitting, tokenization and part-of-speech (POS) tagging must normally all be carried out prior to running a syntactic parser. Syntactic parse results may then be used to carry out more complex tasks, e.g., relation or event extraction, semantic search, etc.

New types of applications can be built more easily if tools that carry out basic processes (sentence splitting, tokenization, etc.) can be reused. Accordingly, large numbers of LRs that perform such processes will be made available on META-SHARE. Although making a library of such LRs readily available from a single point of access is certainly beneficial for developers, it does not necessarily follow that the resources can be reused easily. For example, different LRs can be implemented in different programming languages, have different output formats, or use different data types. This means that, potentially, a large amount of work may be required in order to integrate a set of disparate LRs into a complex application.

As a consequence of these integration problems, new applications that are built may have sub-optimal performance. Often, a particular application can be built using various combinations of component tools. For example, for any given language, there may be a range of tokenizers, POS taggers, etc. Combining different tools in different ways may result in different levels of overall performance of the complete application. However, if a large amount of extra code must be written to facilitate the integration of each possible tool, then performing experiments using different combinations of tools may simply not be an option.

One of the major goals of META-SHARE is to overcome such potential problems of resource reusability, by promoting the use of widely acceptable standards for language resource building, in order to ensure the maximum possible interoperability of language resources (LR). Interoperability between LRs can be achieved in a number of ways. In METANET4U, one of the goals of the project is to demonstrate explicitly the advantages that can be brought to NLP application building by making resources interoperable.

Our work is focussed on the use of the Unstructured Information Management Architecture (UIMA) framework and the associated U-Compare platform. The former constitutes a framework by which LRs can be made into interoperable components, whilst latter provides a graphical user interface which allows interoperable UIMA components to be combined together into workflows using simple drag-and-drop actions, with no coding effort required. Thus, by making LRs available as 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. Once components have been made available as UIMA components, the use of U-Compare allows prototype applications to be built without writing any additional program code. This means that experimentation can be carried out not only by experienced system developers, but also by less technical users, whose programming skills may be limited.

A certain amount of effort is required to convert existing LRs into interoperable UIMA components, requiring some extra program code to be written. As part of the METANET4U project, we are working on writing such code, in order to make a range of our LRs available as UIMA components. These components will include both monolingual LRs operating on various different languages, as well as multi-lingual and speech-based LRs. Using U-Compare, we are then combining the UIMA components in various ways into 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 to allow workflows to be constructed rapidly. This will provide evidence that a more wide-scale adoption in META-SHARE could be advantageous.

In this *Deliverable 4.4 First Version of Pilot Applications*, we describe the first phase of the implementation of the work described above, which consists of a first set of UIMA components, together with a sample set of working workflows that make use of these components. The work follows on from a planning and design phase, which was documented in *Deliverable D2.2 Specification of pilot services and applications.* In D2.2, we determined a set of LRs that were deemed suitable to be made available as UIMA components for use in NLP workflows. We also designed a set of possible workflows that could be built using various combinations of these UIMA components.

At this point, it should be noted that, although the work being carried out on interoperability in METANET4U is concerned only with UIMA /U-Compare, the partner UPF is involved with the PANACEA project[1], which also places a great emphasis on interoperability. The main objective of PANACEA is to build a factory of language resources that automates the

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

stages involved in the acquisition, production, updating and maintenance of language resources required by MT systems and other applications based on language technologies. Such automation is achieved through the implementation of a number of workflows, based on web services. Thus, in order to demonstrate that interoperability can be achieved in different ways, PANACEA as well as UIMA/U-Compare workflows will be made available in META-SHARE. More details about PANACEA, together with sample workflows, were provided in *Deliverable D2.2 Specification of pilot services and applications.*

The remainder of this deliverable is organised as follows. In section 2, very brief information is provided about UIMA components (more detailed information was provided in D2.2). In section 3, relevant details about U-Compare are provided, which build upon the information that was provided in D2.2, providing greater detail regarding the construction of workflows, the display of results, and the U-Compare type system, which helps to further facilitate interoperability of components. These details help to put into context the work reported in the remainder of this deliverable. In section 4, we examine the overall plan of work on UIMA/U-Compare in the METANET4U project, providing greater detail about what has been achieved so far, and what has yet to be completed in the remainder of the project. In section 5, we provide details regarding the LRs that the different partners of the project have already made available as UIMA components. In section 6, we illustrate the workflows that can be constructed using the UIMA components described in section 5. Finally, in section 7, we look ahead to the second phase of the workflow implementation.

2 Creation of UIMA components

The UIMA framework (Ferrucci, et al., 2006) provides a means to make LRs interoperable. Interoperability is achieved by imposing a standard means of communication between the LRs, thus facilitating their easy combination into workflows. For pre-existing LRs, this normally entails the production of some “wrapper” code, whose purpose is to ensure that the LR obtains its input and produces output according to the requirements imposed 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. Thus, the main purposes of the wrapper code are as follows:

a) To obtain from the CAS the information required as input to the LR, and to convert this information into the format required to run the LR.
b) To convert the output of the LR to CAS annotations, and to write these to the CAS.

Once wrapped as UIMA components, the original programming language and input/output formats of the LRs become irrelevant.

In addition to the wrapper code, each UIMA component must be accompanied by an XML file called a *Descriptor*. The descriptor file includes details such as the name of the component, a brief description of the functionality, details of the input/output annotations, parameters that are required in the configuration of the component (such as the location of external files), etc.

3 U-Compare

U-Compare (Kano et al., 2009; Kano et al., 2011) is built on top of UIMA, and its main aim is to support the rapid and flexible construction of language technology (LT) applications from reusable resources, and to allow easy evaluation of such applications against gold-standard annotated data.

3.1 Constructing workflows

U-Compare's graphical user interface makes it easy for users to construct LT applications using simple drag-and-drop actions. To create a workflow, resources are dragged from a library of available components onto a workflow "canvas", in the required order of execution. Once a complete workflow has been specified, it can be run at the click of a button.

The main U-Compare interface window is shown in Figure 1. On the right hand side of the interface is a library of LRs that are available as UIMA components. On the left of the interface is the workflow canvas. To create workflows, the user drags components from the library onto the workflow canvas.

The workflow canvas consists of two sections. At the top is the *Collection Reader* section. The user should drag into this section a component that reads in texts that will be analysed by subsequent components in the workflow. In the example in Figure 1, an *Input Text Reader* has been dragged into this section. This allows the user to type or paste text directly into a text box. Another type of collection readers reads a directory of from the user's file system and processes each file in the directory according to the workflow specification.

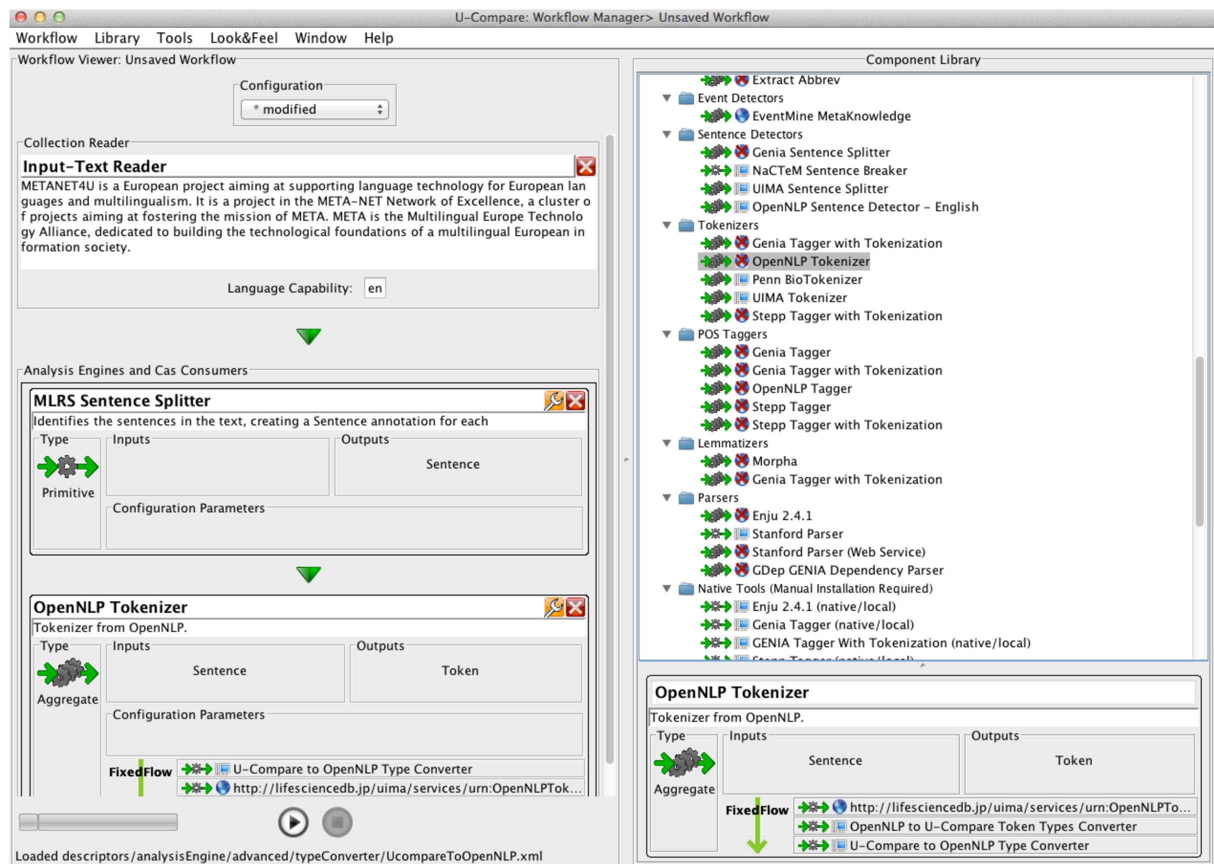


Figure 1: U-Compare interface

The second section of the canvas is the *Analysis Engines and Cas Consumers* section. Into this section, users should drag components that carry out the processing of the text(s) that have been read by the collection reader. In Figure 1, we see that 2 components have been dragged onto this section of the canvas, i.e., a sentence splitter and a tokenizer. These tools will be run on the input text in the order that they appear on the canvas. Each component in this section will generally add new annotations to the CAS. It can be noted that the graphical representation of each component in the workflow shows the types of annotations that are required as input (i.e., those types of annotations that must be present in the CAS in order for the component to run) and those annotations types that are produced as output (i.e., they are added to the CAS following the execution of the component). These input and output types (as well as other details in the graphical representation of the component) are generated from the component's XML descriptor file.

In the case of the *MLRS Sentence Splitter,* the *Inputs* field of its description is blank, meaning that an unannotated text is accepted as input. In the *Outputs* field, one type of annotation is specified, i.e., *Sentence.* For the *OpenNLP Tokenizer* component, *Sentence* annotations are required as input and *Token* annotations are output to the CAS. In

order to build a workflow, it must be ensured that any annotation types that are required as input to a particular component in the workflow are already in the CAS at the time of its execution. The *Inputs* and *Outputs* fields in the graphical representations of the components in U-Compare make it easy to ensure that such requirements are met. For example, in the workflow in Figure 1, the fact that the *MLRS Sentence Splitter* is executed prior to the *OpenNLP Tokenizer* in the workflow means that the *Sentence* annotations required by the tokenizer will already be present in the CAS.

A final point to note about the interface in Figure 1 is the fact that clicking over a component in the library on the right hand side of the interface causes the specification of the component (including its input and output annotation types) to be displayed at the bottom of the right hand pane. In this way, users can check the specification of individual components before adding them to the workflow. Once built, workflows can be saved for later use, by selecting an item in the *Workflow* menu at the top of the window. This is particularly useful when a workflow is complex, and consists of a large number of different processing steps.

3.2 Executing workflows and viewing results

Clicking on the button with the "Play" icon at the bottom of the left hand side of the window shown in Figure 1 causes the workflow to be executed, by reading each of the texts read specified in the *Collection Reader* component, and applying to these texts the tools specified in the *Analysis Engines and Cas Consumers* section. After all processing is complete, a new window appears that allows the annotations added during the workflow to be viewed. The results obtained from the workflow built in Figure 1 are shown in Figure 2.

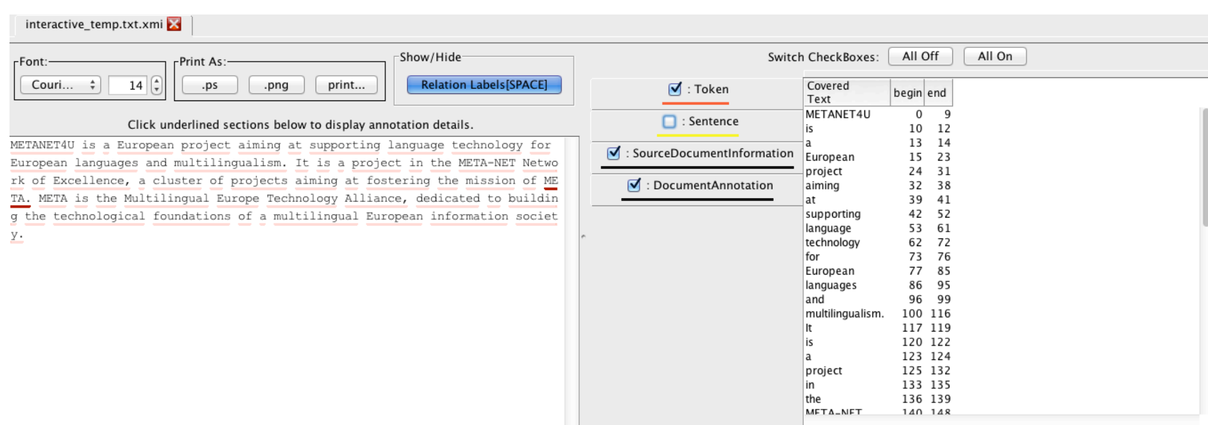


Figure 2: U-Compare annotation viewer

The analysed text is shown on the left-hand side of the window. In the middle of the window, the different types of annotations added during the execution of the workflow are shown. Each type of annotation is shown

using different coloured underlines in the text. Each annotation type has a checkbox next to it, which allows its associated annotations to be "switched" on or off in the view of the text. This feature can be especially useful when a workflow adds many types of annotations to a text, in order to be able to concentrate on specific types of annotations. In Figure 2, only *Token* annotations are displayed. On the right hand side of the window, the features of the annotations are shown in a tabular format. In the case of *Token* annotations, only the covered text, and the beginning and end offsets of the annotations are shown. Other types of annotations will have further attributes, e.g., the POS tag assigned in the case that a POS tagger has also been run. In addition to the annotations themselves, additional details are available regarding the performance of workflows, such as the amount of time taken to execute the components. The speed of execution time can also be an important consideration; the choice of an optimal workflow may involve a trade-off between the quality of the annotations and the time taken to complete the workflow.

U-Compare also provides special facilities that allow the performance of workflows to be evaluated against gold standard data. Given that a number of tools are often available for similar purposes, U-Compare makes it possible to apply several different workflows to a corpus in parallel, and to compare which workflow produces the best results against a gold standard corpus. Such gold standard corpora can be made available as a special type of collection reader component, and indeed, a number of such corpora are planned to be made available in the second release of UIMA components, at M18. Therefore, these features of U-Compare will be described more fully in *Deliverable D4.6 Second version of pilot applications*, due at M18.

3.3 U-Compare type system

Another resource that comes packaged with U-Compare is the *U-Compare type system*. As described above, all annotations that are added to the UIMA CAS have types, e.g., *Sentence, Token,* etc., and, in order to connect components together into workflows, they must "understand" the types of annotations produced by other components in a workflow. Since UIMA itself does not define a set or system of types, UIMA components produced by different developers may use their own system of annotation types, which can cause problems of interoperability for UIMA components developed by different groups. Whilst achieving consensus on a common type system suitable for encoding the inputs/outputs of all possible LRs would be problematic, U-Compare defines a "sharable" system of types, that includes syntactic, semantic and document-level annotation types that are commonly produced by NLP applications. The idea is that all components available in U-Compare should produce annotations that are compatible with this type system. The U-Compare type system consists of fairly general types, which may be specialised if necessary using subtypes.

By ensuring that as many UIMA-wrapped LRs as possible comply with the U-Compare type system, it can be ensured that compatibility between many LRs developed by different groups can be achieved, at least at an intermediate level of the type hierarchy.

The utility of U-Compare and its type system have already been demonstrated through its existing library of over 50 components (some of which are described below, as those that will also be made available in META-SHARE), each of which is compatible with the U-Compare type system. This existing library is mainly focussed on the processing of biomedical texts in English.

In order to ensure the interoperability of the new UIMA components being created during METANET4U, the annotation types that they will use to encode their inputs and outputs will comply with the U-Compare type system as much as possible. As reported in D2.2, part of our initial analysis of the input and output of the LRs that we selected showed that the existing types appeared to be largely sufficient for the new components. However, where necessary, the type system will be expanded to accommodate data types that were not dealt with by the original system, such as speech based input/output. Our goal is to produce a type system that can be employed regardless of LR language or type.

4 Work on UIMA components and U-Compare workflows in METANET4U

In order to put into context the work described in this deliverable, we provide in this section a brief overview of the plan of work regarding the development of UIMA components and U-Compare workflows, including details of what has already been completed, and what is yet to be done. The main tasks to be carried out on this work during the lifetime of the project consist of the following:

1) **Identifying a set of LRs to wrap 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*. To these were added some additional resources, in order to better demonstrate the potential for multi-lingual and speech-based applications within U-Compare.
2) **Designing a set of workflows that make use of the UIMA-wrapped components.** The wrapped components can be combined flexibly into workflows. In order to showcase this flexibility and versatility, a number of specific workflows that make use of the wrapped components have been designed. These workflows have the following purposes:

   a. They demonstrate that different components can be combined and reused to carry out several different tasks.
   b. They demonstrate that components developed by different partners and using different implementation methods can be combined seamlessly into workflows.
   c. 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. Code has to be written to ensure that input/output of the selected LRs is handled in the way required by UIMA. This stage has been partially completed. This stage also identifies whether the existing U-Compare type system can handle the input/output types of the components, or whether any expansions to the type system are required. The type system should remain reasonably compact, but yet be able to handle a wide range of different LRs. Since the majority of project partners had not worked with UIMA and U-Compare before, the first phase of the implementation phase was devoted to learning how to carry out the wrapping process, and how to test the wrapped components in U-Compare. In order to facilitate the learning process, UNIMAN produced a short tutorial document, which also provided links to relevant information in the UIMA online documentation. UNIMAN has also supported partners in the creation of their own UIMA components. A one day meeting/workshop was also held in Manchester a few weeks before the delivery of this first set of UIMA components and workflows, in order to consolidate the work carried out in the first few months of the implementation phase, and to provide support to partners with any outstanding implementation issues.
4) **Implementation of the workflows.** Step 2) provides a guide as 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 a large number of 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, one for each language and task. These take the form of special files containing read-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.
5) **Integration into the digital exchange platform.** Both components and workflows will be made available for download on META-SHARE,

allowing them to be imported into U-Compare by users of the platform. It is also planned to integrate many of the components into the "core" U-Compare library at a later stage in the project, so that a range of different components operating in a range of languages is readily available to users. Certain components may still have to be imported separately into U-Compare, according to the compatibility of the terms of their licences with those of U-Compare.

Of the steps outlined above, 1) and 2) were carried out during a planning phase that was undertaken between M2 and M5 of the project. The results of this planning phase are described in detail in D2.2 *Specification of pilot services and applications*. As a brief summary, the planning phase resulted in the identification of a total of 67 resources that would be made available as UIMA components, together with 26 different workflows that would make use of these components.

Steps 3) and 4) constitute the main implementation phase of the project. The wrapped UIMA components and associated workflows are being delivered in 2 stages, the first at M12, and the second at M18. The work required for step 5) will be carried out between M19 and M24.

This *Deliverable 4.4 First Version of Pilot Applications* accompanies the first release of components and workflows, which mainly constitute more basic levels of text processing. Following the first phase of implementation, there are 32 UIMA-wrapped components (including a number that had previously been wrapped by UNIMAN). The components can handle 8 different languages. In terms of workflows, a total of 10 have been (partially) implemented, of which 7 can operate on multiple languages. By "partially", we mean that some of the workflows can currently operate only on a subset of the planned languages, or else they can only be created using a subset of the possible planned variations.

5 Resources wrapped as UIMA components

In this section, we provide details of the 31 resources that are included in the first release of wrapped UIMA components that accompanies this report, of which 15 have been newly wrapped during this first phase of the implementation period (the remainder being resources already wrapped as part of the main U-Compare release). In general, partners have begun by wrapping their simplest components. The motivations for this are two-fold. Firstly, simpler resources are generally easier to wrap as UIMA components than more complex resources and, as explained above, at the beginning of the project, partners generally had no previous experience of carrying out these wrapping activities. Secondly, it means that a set of "core" workflows can already be built at the end of this first phase of implementation, which will form the basis of many of the more complex

workflows that will be implemented during the second phase, between M13 and M18.

As mentioned above, U-Compare comes packaged together with a library of components, which include amongst them some of UNIMAN's components that are described below. LRs that have been newly wrapped during this first implementation phase are currently provided in the form of Java Archive (jar) files, containing the program code and the descriptor file. Using jar files, new components can easily be imported and tested in U-Compare, allowing us to demonstrate their interoperability. The process of adding new components packaged as jar files to the U-Compare library is illustrated in Figure 3.

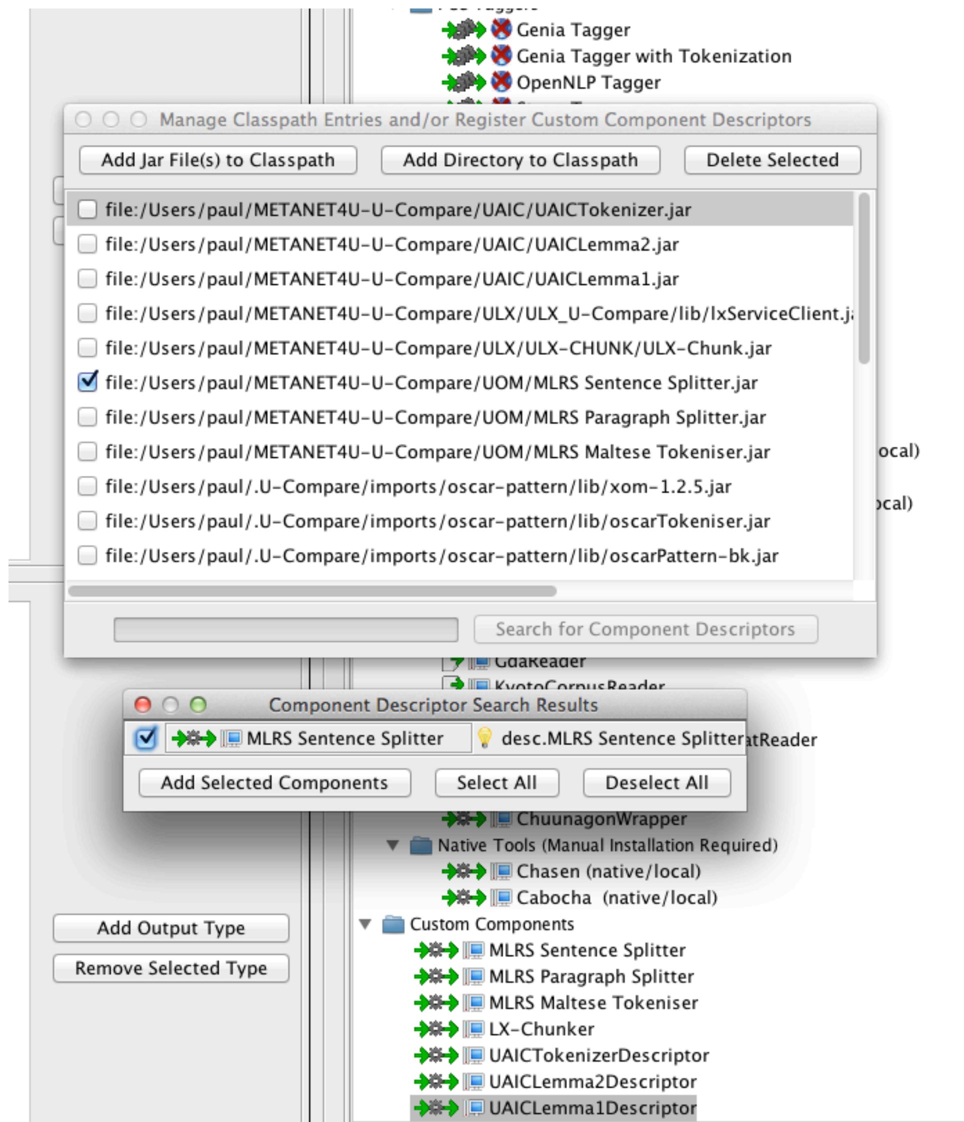


Figure 3: Adding new components to the U-Compare library

Choosing a menu item on the U-Compare interface causes a window with a list of jar files containing currently imported components to be displayed, as shown in the centre of Figure 3. New jar files can be added

to this list by clicking on the "Add Jar File(s) to classpath" button. New components are added to the library by checking the boxes next to the appropriate jar files and then clicking on the button labelled "Search for Component Descriptors" is clicked. This causes descriptor file(s) in the selected jar files to be found. The search results are displayed in a separate window, with the option to add the components associated with the checked descriptors to the U-Compare library. Clicking on the "Add Selected Components" button causes the new components to appear in the component library pane, under the "Custom Components". Once added, components can be moved from this section to other categories in the library, to make them easier to locate.

In the remainder of this section, we provide details of the 32 resources that are currently wrapped as UIMA components. The resources are grouped according to the contributing partner, and the following details are provided:

- 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.

The majority of the components listed for UNIMAN are already part of U-Compare, with the exception of two newly wrapped components (i.e., the Apertium morphological analyser and the Apertium tagger). These newly wrapped resources, together with all other resources that have been newly wrapped by other partners, 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

The list of components will be augmented with further new components during the second phase of the implementation period.

5.1 University of Lisbon (ULX)

Tools

LX-Chunker

Description: Portuguese sentence and paragraph boundary detector. It unwraps sentences split over different lines.
Languages covered: Portuguese
Original resource implementation: Web service
Input: Plain text
U-Compare input type: N/A
Output: Paragraphs, Sentences
U-Compare output type:
`org.u_compare.shared.document.text.Paragraph,`
`org.u_compare.shared.syntactic.Sentence`
U-Compare type details: These are the originally planned output types. The existing U-compare types are sufficient for the output of this tool, without any need for extension.
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 on the U-Compare claspath. 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.
NOTE: Whilst this component is correcly wrapped, and works perfectly when U-Compare is started from within the Eclipse programming environment (which partners have been encouraged to use to develop their code), it currently does not work when U-Compare is started independently of Eclipse. It has been discovered that this is due to bug in U-Compare concerning the reading of lxServiceClient.jar, which is not trivial to rectify. Thus, for the time being, we have not included this component in any importable workflows. However, we still include the component as part of this release, as it is a fully functional UIMA component, and its functionality can be tested by starting U-Compare from within Eclipse. The U-Compare bug will be fixed as soon as possible during the second phase of the implementation.

LX-Tokenizer

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:
`org.u_compare.shared.syntactic.Token`
U-Compare type details: These are the originally planned input and output types. However, the information output by the current UIMA wrapped component does not include all of the information output by the original tool, which outputs richer information than only basic tokens. Some examples of this extended functionality include expanding contractions and detaching clitic pronouns from verbs. Such information would have to be stored as additional attributes of the token, but `org.u_compare.shared.syntactic.Token` does not have provision for storing such extra information. However, it is planned, during the second phase of the implementation, to make changes to this component. The specific plan is to extend the `org.u_compare.shared.syntactic.Token` type, to allow at least some of the extra information produced by the LX-Tokenizer tool to be stored as additional attributes. Some of this information is required as input to the LX-Tagger tool, which will be wrapped during the second phase of implementation. Thus, in order to be able to create workflows that include the LX-Tagger, the extra information produced by the LX-Tokenizer must be stored within the UIMA CAS.
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.
NOTE: Whilst this component is correcly wrapped, and works perfectly when U-Compare is started from within the Eclipse programming environment (which partners have been encouraged to use to develop their code), it currently does not work when U-Compare is started independently of Eclipse. This is due to the same bug in U-Compare that was described for the LX-Chunker tool, described above. Therefore, in the same way as for that tool, we do not currently include it in any importable workflows, but still provide the jar file for the individual component.

5.2 IST – Instituto Superior Técnico

No components have been wrapped yet. The E-TXT2DB will be wrapped a U-Compare component during the second phase of the implementation.

5.3 University of Manchester – UNIMAN

Tools

NEMine

Description: Detects gene and protein names in text
Languages covered: English
Input: Sentences
Original resource implementation: Web service
U-Compare input type: org.u_compare.shared.syntactic.Sentence
Output: Protein and Gene named entities
U-Compare output type:
uk.ac.nactem.nemine.NeMineProtein [subtype of org.u_compare.shared.semantic.bio.protein]
uk.ac.nactem.nemine.NeMineGene [subtype of org.u_compare.shared.semantic.bio.gene]
U-Compare type details: The output types are subtypes of existing types in the U-Compare type system. These are retained for historical reasons but do not affect the interoperability of the tool with other components.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

STEPP tagger (with tokenization) (previously wrapped)

Description: POS tagger, tuned to biomedical text
Languages covered: English
Original resource implementation: Web service
Input: Sentences
U-Compare input type: org.u_compare.shared.syntactic.Sentence
Output: POS tagged tokens
U-Compare output type:
jp.ac.u_tokyo.s.is.www_tsujii.tools.stepptagger.SteppToken
[subtype of org.u_compare.shared.syntactic.POSToken]

U-Compare type details: The output type is a subtype of an existing type in the U-Compare type system, as indicated. The type used is retained for historical reasons but does not affect the interoperability of the tool with other components.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

STEPP tagger (no tokenization) (previously wrapped)

Description: POS tagger, tuned to biomedical text
Languages covered: English
Original resource implementation: Web service
Input: Tokens
U-Compare input type: `org.u_compare.shared.syntactic.Token`
Output: POS tagged tokens
U-Compare output type:
`jp.ac.u_tokyo.s.is.www_tsujii.tools.stepptagger.SteppToken`
[subtype of `org.u_compare.shared.syntactic.POSToken`]
U-Compare type details: The output type is a subtype of an existing type in the U-Compare type system, as indicated. The type used is retained for historical reasons but does not affect the interoperability of the tool with other components.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

NaCTeM sentence breaker (previously wrapped)

Description: Detects sentence boundaries using heuristic rules
Languages covered: English
Original resource implementation: Java application
Input: Plain text
U-Compare input type: N/A
Output: Sentences
U-Compare output type: `org.u_compare.shared.syntactic.Sentence`
U-Compare type details: The core U-compare type is sufficient for the output of this tool, without any need for extension.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

GENIA tagger (with tokenization) (previously wrapped)

Description: Performs tokenisation, POS tagging, lemmatization, syntactic chunking and named entity recognition. Tuned for biomedical text
Languages covered: English
Original resource implementation: Web service
Input: Sentences
U-Compare input type: `org.u_compare.shared.syntactic.Sentence`
Output: Tokens with pos tags and base forms, syntactic chunks, named entities
U-Compare output type:
`jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger.GeniaToken` [subtype of `org.u_compare.shared.syntactic.RichToken`], `jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger.GeniaProtein` [subtype of `org.u_compare.shared.semantic.bio.protein`], `jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger.GeniaRNA` [subtype of `org.u_compare.shared.semantic.bio.RNA`], `jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger.GeniaCellLine` [subtype of `org.u_compare.shared.semantic.bio.CellLine`] `jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger.GeniaCellType` [subtype of `org.u_compare.shared.semantic.bio.CellType]`
U-Compare type details: The output types are subtypes of existing types in the U-Compare type system. These are retained for historical reasons but do not affect the interoperability of the tool with other components.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

GENIA tagger (no tokenization) (previously wrapped)

Description: Performs tokenization, POS tagging, lemmatization, syntactic chunking and named entity recognition. Tuned for biomedical text
Languages covered: English
Original resource implementation: Web service
Input: Tokens
U-Compare input type: org.u_compare.shared.syntactic.Token
Output: Tokens with pos tags and base forms, syntactic chunks, named entities
U-Compare output type:
`jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger.GeniaToken` [subtype of `org.u_compare.shared.syntactic.RichToken`], `jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger.GeniaProtein` [subtype of `org.u_compare.shared.semantic.bio.protein`],

jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger.GeniaRNA [subtype of org.u_compare.shared.semantic.bio.RNA], jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger.GeniaCellLine [subtype of org.u_compare.shared.semantic.bio.CellLine] jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger.GeniaCellType [subtype of org.u_compare.shared.semantic.bio.CellType]
U-Compare type details: The output types are subtypes of existing types in the U-Compare type system. These are retained for historical reasons but do not affect the interoperability of the tool with other components.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

GENIA Sentence detector (previously wrapped)

Description: Performs sentence splitting, tuned for biomedical text
Languages covered: English
Input: Plain text
Original resource implementation: Web service
U-Compare input type: N/A
Output: Sentences
U-Compare output type: jp.ac.u_tokyo.s.is.www_tsujii.tools.geniass.Sentence [subtype of org.u_compare.shared.syntactic.Sentence]
U-Compare type details: The output type is a subtype of an existing type in the U-Compare type system, as indicated. The type used is retained for historical reasons but does not affect the interoperability of the tool with other components.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

NaCTeM Species word detector (previously wrapped)

Description: Detects words that indicate model organisms (e.g. mouse, human)
Original resource implementation: Web service
Languages covered: English
Input: Sentence split and part-of-speech tagged text
U-Compare input type: org.u_compare.shared.syntactic.Sentence, org.u_compare.shared.syntactic.POSToken
Output: Species tagged words
U-Compare output type: uk.ac.nactem.semantic.bio.Species [new subtype of org.u_compare.shared.semantic.NamedEntity]

U-Compare type details: The type `org.u_compare.shared.semantic.NamedEntity` includes several subtypes corresponding to named entities. However, “species” was not amongst them. Hence a new subtype was created for this tool. The extended type allows the id of the species (according to the NCBI taxonomy of model organisms) to be stored.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

ExtractAbbrev (previoiusly wrapped)

Description: Extracts abbreviations and their definitions from biomedical text
Languages covered: English
Original resource implementation: Web service
Input: Sentence split and tokenized text, with parts-of-speech assigned
U-Compare input type: `org.u_compare.shared.syntactic.Sentence`, `org.u_compare.shared.syntactic.POSToken`
Output: abbreviations
U-Compare output type: `uk.ac.nactem.semantic.Abbreviation` [subtype of `org.u_compare.shared.semantic.NamedEntity`]
U-Compare type details: The `org.u_compare.shared.semantic.NamedEntity` type of the U-compare type system was extended to handle the output of this tool, i.e, a specific type of named entity corresponding to abbreviations.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

NaCTeM Species Disambiguator (previously wrapped)

Description: Normalises biological named entity mentions in text to NCBI Taxomony IDs, which indicate the entities' model organisms
Languages covered: English
Original resource implementation: Web service
Input: Sentence split text, with part of speech and morophlogically analysed tokens, named entities corresponding to proteins, genes and RNA and abbreviations
U-Compare input type: `org.u_compare.shared.syntactic.Sentence`, `org.u_compare.shared.syntactic.RichToken`, `org.u_compare.shared.semantic.bio.Protein`, `org.u_compare.shared.semantic.bio.Gene`, `org.u_compare.shared.semantic.bio.RNA`, `uk.ac.nactem.semantic.Abbreviation` [subtype of `org.u_compare.shared.semantic.NamedEntity`]
Output: Normalised biological entities

U-Compare output type: `uk.ac.nactem.semantic.bio.SpeciesNormalizedEntity` [subtype of `org.u_compare.shared.semantic.NormalizedEntity`]
U-Compare type details: The `org.u_compare.shared.semantic.NormalizedEntity` type of the U-compare type system was extended to be specialised for normalised entities that describe species.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

<u>Enju Parser</u> (previously wrapped)

Description: HPSG parser
Languages covered: English
Original resource implementation: Web service
Input: Sentence split and tokenized text, with parts-of-speech assigned
U-Compare input type: `org.u_compare.shared.syntactic.Sentence`
Output: Tokens, Sentences, Constituents
U-Compare output type:
`jp.ac.u_tokyo.s.is.www_tsujii.tools.enju.EnjuToken` [subtype of `org.u_compare.shared.syntactic.Token`]
`jp.ac.u_tokyo.s.is.www_tsujii.tools.enju.EnjuSentence` [subtype of `org.u_compare.shared.syntactic.Sentence`]
`jp.ac.u_tokyo.s.is.www_tsujii.tools.enju.EnjuConstituent` [subtype of `org.u_compare.shared.syntactic.Constituent`]
U-Compare type details: The output types are subtypes of existing types in the U-Compare type system, as indicated. The types used are retained for historical reasons but do not affect the interoperability of the tool with other components.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

<u>OpenNLP Sentence detector</u> (previously wrapped)

Description: Detects sentences in plain text
Languages covered: English
Original resource implementation: The sentence detector had previously been wrapped as a UIMA component, but using the OpenNLP Wrapper Type System. Thus, type conversion was required to comply with the U-Compare type system.
Input: Plain text
U-Compare input type: N/A
Output: Sentences
U-Compare output type:
`org.u_compare.shared.syntactic.Sentence.`

U-Compare type details: This original U-Compare type is sufficient to encode the output of this tool.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

OpenNLP Tokenizer (previously wrapped)

Description: Splits sentences into tokens
Languages covered: English
Original resource implementation: The tokenizer had previously been wrapped as a UIMA component, but using the OpenNLP Wrapper Type System. Thus, type conversion was required to comply with the U-Compare type system.
Input: Plain text
U-Compare input type: `org.u_compare.shared.syntactic.Sentence`
Output: Sentences
U-Compare output type:
`jp.ac.u_tokyo.s.is.www_tsujii.tools.opennlp.Token` [subtype of `org.u_compare.shared.syntactic.Token`]
U-Compare type details: The output type is a subtype of an existing type in the U-Compare type system, as indicated. The type used is retained for historical reasons but does not affect the interoperability of the tool with other components.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

OpenNLP POS tagger (previously wrapped)

Description: Performs POS tagging
Languages covered: English
Original resource implementation: The tagger had previously been wrapped as a UIMA component, but using the OpenNLP Wrapper Type System. Thus, type conversion was required to comply with the U-Compare type system
Input: Sentences, tokens
U-Compare input type: `org.u_compare.shared.syntactic.Sentence`, `org.u_compare.shared.syntactic.Token`
Output: Tokens with pos tags and base forms
U-Compare output type:
`jp.ac.u_tokyo.s.is.www_tsujii.tools.opennlp.POSToken` [subtype of `org.u_compare.shared.syntactic.POSToken`].
U-Compare type details: The output type is a subtype of an existing type in the U-Compare type system, as indicated. The type used is

retained for historical reasons but does not affect the interoperability of the tool with other components.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

Morpha (previously wrapped)

Description: Returns lemma and inflection type of a word, given part of speech
Languages covered: English
Original resource implementation: Web service
Input: Sentences, tokens with parts-of speech attached
U-Compare input type: `org.u_compare.shared.syntactic.Sentence`, `org.u_compare.shared.syntactic.POSToken`
Output: Tokens with morphological analyses attached
U-Compare output type: `org.u_compare.shared.syntactic.RichToken`
U-Compare type details: This original U-Compare type is sufficient to encode the output of this tool.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project.

Stanford Parser (previously wrapped)

Description: Syntactic parser
Languages covered: English
Original resource implementation: Web service
Input: Sentences
U-Compare input type: `org.u_compare.shared.syntactic.Sentence`
Output: POS tags and dependency parsing
U-Compare output type: `org.u_compare.shared.syntactic.StanfordDependency`, `org.u_compare.shared.syntactic.POSToken`
U-Compare type details: These original U-Compare types are sufficient to encode the output of this tool.
Implementation details/issues: Not applicable. Tool wrapped as a U-Compare component prior to the start of the project

Apertium Morphological Analyser

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, Romanian

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.
Implementation details/issues: This is one module of the Apertium machine translation system (third-party, open-source software). In D2.2, we originally envisaged one component that would perform both morphological analysis and part-of-speech tagging. However, in the implementation, we have followed the modular structure of Apertium, and created two separate components. The wrapped component can operate on any language for which the linguistic data file is available for download from Apertium. The languages listed above are those which are relevant to the countries involved in METANET4U, although several other languages are available. The analysis language is determined according to the data file provided as an argument. In the UIMA component, this is determined simply by setting a parameter. Whilst the UIMA component is fully operational, we have found during the wrapping process several bugs in the Java port of Apertium. Such bugs include the inability to handle slashes or blank lines in the input. Our wrapped UIMA component catches and handles these errors. In the second phase of the implementation, we will test the module more thoroughly to more fully determine and handle further bugs in the Java version of Apertium.

Apertium Tagger

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: 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.
Implementation details/issues: This is one module of the Apertium machine translation system (third-party, open-source software). The wrapped component can operate on any language for which the linguistic data file is available is Apertium. The languages listed above are those which are relevant to the countries involved in METANET4U, although several other languages are available. The analysis language is determined according to the data file provided as an argument. In the UIMA component, this is determined simply by setting a parameter. A further parameter allows tagger options to be specified.

5.4 University Alexandru Ioan Cuza (UAIC)

Tools

Tokenizer-UAIC

Description: Basic tokenizer
Languages covered: Most European languages (may have trouble with Spanish and unusual diacritics).
NOTE: This tool thus has wider coverage than was originally reported in D2.2, where it was stated that the tool only works for English and Romanian.
Original resource implementation: Perl program
Input: Plain text
U-Compare input type: N/A
Output: Tokens
U-Compare output type:
`org.u_compare.shared.syntactic.Sentence`.
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: The original Perl program was re-implemented in Java, to facilitate easier wrapping as a UIMA component.

Lemmatizer-UAIC-v1

Description: Determines base form for tokens. The performance is only slightly inferior (approximately 99% as accurate) to Lemmatizer-UAIC-v2 (which takes POS-tagged input).
Languages covered: Romanian, UTF-8 encoding with diacritics.
Original resource implementation: Java application
Input: POS tagged, tokenized text
U-Compare input type: `org.u_compare.shared.syntactic.Token`
Output: Lemmatized tokens
U-Compare output type:
`org.u_compare.shared.syntactic.RichToken`
U-Compare type details: These are the originally planned input/output types. The existing U-Compare types are sufficient for the input/output of this tool, without any need for extension.
Implementation details/issues: The original Perl program was re-implemented in Java, to facilitate easier wrapping as a UIMA component, given that Java is the most straightforward language to use to perform the wrapping.

Lemmatizer-UAIC-v2

Description: Determines base form for tokens, using POS-tagged tokens as input. This is the lemmatizer tool described in D.2.2.
Languages covered: Romanian, UTF-8 encoding with diacritics.
Original resource implementation: Java application
Input: POS tagged, tokenized text
U-Compare input type: `org.u_compare.shared.syntactic.POSToken`
Output: Lemmatized tokens
U-Compare output type:
`org.u_compare.shared.syntactic.RichToken`
U-Compare type details: These are the originally planned input/output types. The existing U-Compare types are sufficient for the input/output of this tool, without any need for extension.
Implementation details/issues: The original Perl program was re-implemented in Java, to facilitate easier wrapping as a UIMA component.

5.5 RACAI – Romanian Academy

Tools

The tool components being made available by RACAI are built based on TTL, a Perl module that performs sentence splitting, tokenization, POS

tagging, lemmatization and chunking (shallow parsing without phrase attachment and no recursive structures). It is largely described in Ion (2007). To be readily used in other programming languages for different NLP applications, TTL has been wrapped as a SOAP web service hosted by the Apache httpd web server (http://httpd.apache.org/). The WSDL of this web service is located at http://ws.racai.ro/ttlws.wsdl and the whole enterprise is described in (Tufiș et al., 2008). We implemented an Annotation Engine (AE in UIMA's terminology) for each operation that is exported by the TTL web service: sentence splitting and tokenization (bundled because the sentence splitter offers some information to the tokenizer that cannot be ignored/lost), POS tagging, lemmatization and chunking.

In all cases, the language of analysis is determined by the setting the "language" attribute, which is available in U-Compare's "Input Text Reader" and "File System Collection Reader", as either "en", "ro" or "fr", as all components can operate in English, Romanian or French.

The typical workflow using the TTL's UIMA components is: TTL-Tokenizer, TTL-POSTagger, TTL-Lemmatizer and TTL-Chunker. The chain may be interrupted at any point with the condition that the order and the progression of the elements is not changed.

TTL-Tokenizer

Description: Performs tokenization and sentence splitting.
NOTE: In D2.2, the functionality of this component was stated only as tokenization. However, sentence splitting functionality is also included. This is because the tokenizer requires some information about sentences in order to run correctly. Additionally, contrary to what was stated in D2.2, the input is plain text, rather than sentence annotations.
Languages covered: Romanian, English, French
Original resource implementation: Perl
Input: Plain Text
U-Compare input type: `N/A`
Output: Tokens and sentence
U-Compare output type: `org.u_compare.shared.syntactic.Sentence ,org.u_compare.shared.syntactic.RichToken`
U-Compare type details: As mentioned above, sentences as well as tokens are output. In the case of tokens, the `RichToken` is used, because in some cases, the POS and lemma can be determined prior to running the POS tagger. These are stored in the `posString` and `base` attributes, respectively.
Implementation details/issues: UIMA wrapper around the web service, as described above. The tool uses precompiled lists of abbreviations (e.g. "etc.", "i.e.", "b.c." and so on) and multi-words expressions (e.g. "that is",

"as long as", "as soon as", etc.), so as not to break textual units at inappropriate places.

TTL-Tagger

Description: Part-of-speech tagger
Languages covered: Romanian, English, French
Original resource implementation: Perl
Input: Sentences, tokens
U-Compare input type: `org.u_compare.shared.syntactic.Sentence,org.u_compare.shared.syntactic.RichToken`
Output: POS tagged tokens
U-Compare output type: `org.u_compare.shared.syntactic.RichToken`
U-Compare type details: The `RichToken`s used as input may already have certain POS and lemmas assigned by the TTL-Tokenizer. This component updates the `RichToken`s so that each has a part-of-speech assigned, in the `posString` attribute.
Implementation details/issues: UIMA wrapper around the web service, as described above. POS tagging follows he HMM model described by Brants (2000) but modified in order to improve the tagging of unknown words.

TTL-Lemmatizer

Description: Finds base forms of tokens
Languages covered: Romanian, English, French
Original resource implementation: Perl
Input: POS-tagged tokens
U-Compare input type: `org.u_compare.shared.syntactic.Sentence,org.u_compare.shared.syntactic.RichToken`
Output: Tokens with base forms added
U-Compare output type: `org.u_compare.shared.syntactic.RichToken`
U-Compare type details: The `RichToken`s used as input may already have certain POS and lemmas assigned by the TTL-Tokenizer. This component updates the `RichToken`s so that each has a lemma assigned, in the `base` attribute.
Implementation details/issues: UIMA wrapper around the web service, as described above. Lemmatization is carried out through the use of an existing lexicon which stores for each word form, its lemma and its POS tag. When the word cannot be found in the dictionary, a list of automatically learned rules that transform from the word form to lemma comes into play, and a statistical choice is made to select one of the candidate lemmas.

TTL-Chunker

Description: Identifies syntactic chunks
Languages covered: Romanian, English, French
Original resource implementation: Perl
Input: POS-tagged and lemmatized tokens
U-Compare input type: `org.u_compare.shared.syntactic.Sentence,org.u_compare.shared.syntactic.RichToken`
Output: Syntactic chunks
U-Compare output type: `org.u_compare.shared.syntactic.Constituent`
U-Compare type details: These are the originally planned input/output types. The existing U-Compare types are sufficient for the input/output of this tool, without any need for extension.
Implementation details/issues: UIMA wrapper around the web service, as described above. Chunking is carried out using regular expressions defined over sequences of POS tags. Spans of text are thus identified that represent a noun phrase, a verb phrase, a prepositional phrase and an adjectival/adverbial phrase.

5.6 University of Malta (UOM)

Tools

MLRS Paragraph Splitter
Description: Identifies the paragraphs in a given text, creating an annotation for each paragraph.
Languages covered: Language independent
Original resource implementation: Java application
Input: Plain text
U-Compare input type: N/A
Output: Paragraphs
U-Compare output type:
`org.u_compare.shared.document.text.Paragraph`.
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: No problems encountered - straightforward wrapping, given compatibility between original implementation language and UIMA.

MLRS Sentence Splitter
Description: Identifies the sentences in a given text, creating an

annotation for each identified sentence.
Languages covered: Language independent – tweaked to facilitate better performance on Maltese text.
Original resource implementation: Java application
Input: Plain text
U-Compare input type: N/A
Output: Sentences
U-Compare output type:
`org.u_compare.shared.syntactic.Sentence`.
U-Compare type details: This is originally planned output type. The existing U-compare type is sufficient to encode the output of this tool, without any need for extension.
Implementation details/issues: In D2.2, it was specified that this tool required paragraph annotations as input. In fact, there are different versions of the tool. The one that has currently been wrapped as a UIMA component works on raw text. However, an alternative version of the component, that accepts paragraph annotations as input, will also be created and tested during the second phase of the implementation. Experiments will be undertaken to determine whether a workflow consisting of paragraph breaking followed by sentence splitting produces different results to those achieved by applying the sentence splitter directly to raw text. The wrapping of the tool was fairly straightforward, given compatibility between original implementation language and UIMA.

<u>MLRS Maltese Tokenizer</u>
Description: Designed to tokenize Maltese text. In written texts, tokens are:

- orthographic words (i.e. words between spaces)
- elements attached to words between spaces, but separated from them by:
  - an apostrophe
  - a hyphen
- punctuation

Languages covered: Maltese
Original resource implementation: Java application
Input: Plain text
U-Compare input type: N/A
Output: Tokens
U-Compare output type:
`org.u_compare.shared.document.text.Token`.
U-Compare type details: This is originally planned output type. The existing U-Compare type is sufficient to encode the output of this tool, without any need for extension.
Implementation details/issues: In D2.2, it was specified that this tool

required sentence annotations as input. In fact, there are different versions of the tool. The one that has currently been wrapped as a UIMA component works on raw text. However, an alternative version of the component, that accepts as input sentence annotations, will also be created and tested during the second phase of the implementation. Experiments will be undertaken to determine whether a workflow consisting of sentence splitting followed by tokenization produces different results to those achieved by applying the sentence splitter directly to raw text. The wrapping of the tool was fairly straightforward, given compatibility between original implementation language and UIMA.

5.7 UPC - Universitat Politècnica de Catalunya

Tools

N-II translation

Description: Statistical machine translation system
Languages covered: English -> Spanish
Original resource implementation: Web service
Input: Plain text
U-Compare input type: N/A
Output: Paragraphs with translations attached
U-Compare output type: `cat.talp.metanet4u.nii.Translation` [subtype of `org.u_compare.shared.document.text.Paragraph`]
U-Compare type details: The currently used output type extends the U-Compare Document annotation type, since the translation service is handled paragraph by paragraph. The new, extended subtype includes an attribute to store the translated text. The current solution is a temporary one, to allow the component to be tried out in the current version of U-Compare. The envisaged method of implementation for multilingual components is for the source language text and target language text to be treated as different "views" of the document. UIMA provides a mechanism for this, with each being called a "sofa" (subject of analysis). Currently, the graphical annotation viewers provided with U-Compare can only handle a single view of the document. However, work is planned to rectify this during the lifetime of the project (see section 7), so that multiple views of a document can be visualised side by side. Once the new annotation viewer has been implemented, the output of this component will be changed accordingly.
Implementation details/issues: The currently wrapped component deals with one direction of translation, i.e., from English to Spanish. The original web service can, however, deal with translation in the other direction, i.e., from Spanish to English, as well as Spanish to Catalan, and

vice versa. These additional capabilities will be added to an updated version of the component, to be released during the second phase of the implementation. It is currently intended to allow the user to choose the language pair and direction of translation by setting parameters in the configuration of the component. Such configuration can be carried out easily using the U-Compare interface. The component is accompanied by 2 jar files from the Jersey open source project, jersey-bundle-1.11.jar and jsr311-api-1.1.1.jar, which must also be added to the U-Compare classpath in order for the component to work (to allow web services to be called). The component itself works by calling two web services: the first is used to request a translation, in response to which a translation ID is returned. The second web service allows a translation with a given ID to be requested. The component splits the input into paragraphs, requests translations for all the paragraphs, and then requests the translation results in the same order.

5.8 UPF- Universitat Pompeu Fabra

The main involvement of UPF is in the PANACEA workflow system. However, it is hoped in the second phase of the implementation to be able to make available some of their PANACEA web services as U-Compare components, as these provide a basic set of processing tools for the Spanish and Catalan languages, including tokenization, morphological analysis, tagging and parsing. These web services were described in Deliverable 2.2.

6 Workflows

The components that have been wrapped as U-Compare components, as detailed in the last section, can currently be combined together into workflows to perform 10 out of the 26 NLP tasks that were determined in Deliverable 2.2. Several of these workflows can operate on multiple languages and can use various combinations of components developed by different partners. As mentioned above, the workflows that can be constructed at this point generally correspond to simpler tasks, which will often form part of the more complex tasks that will be possible to perform by the end of the second implementation phase.

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 shown, which illustrated all possible paths though these 26 tasks, using the components that are planned to be made available during the METANET4U project, showing 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 various 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 could be used as a guide to the possible workflows that could be built using the components that are being made available in METANET4U.

As mentioned above, particular workflows can be exported from U-Compare as single files ("ucz" or U-Compare zip files). These files contain details of the components that form the workflow, as well as the jar files of any user-imported components. This means that users can import and use the workflows by performing a single operation, even if they have not previously imported all of the components that are used in the workflow. Once imported, workflows appear as items in U-Compare's *Workflow* menu, with the prefix "imported", as shown in Figure 4. Clicking on the workflow name will cause the workflow canvas to be populated with the components in the workflow.

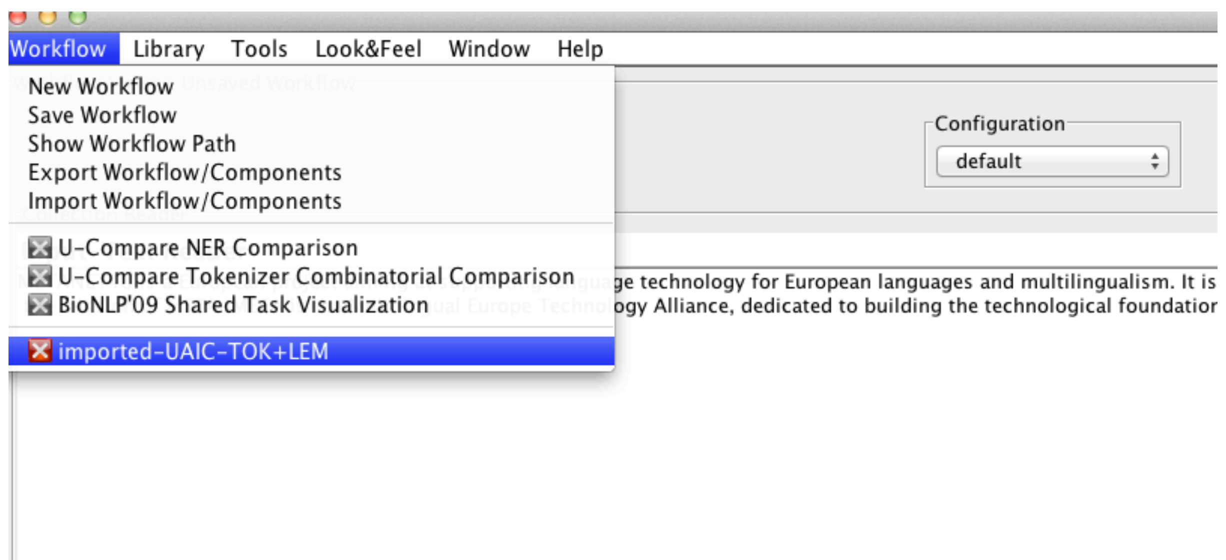


Figure 4 *Workflow* menu showing an imported workflow

As part of this deliverable, we provide a set of sample "implemented" workflows (i.e, ucz files), which can be imported into U-Compare. These are provided in addition to the set of jar files corresponding to individual components, as detailed above. Given the large number of potential alternative workflows that can be created for each NLP task, using the components that are being wrapped during METANET4U, we are not providing an exhaustive set of all possible implemented workflows.

Rather, we include a selection of sample implemented workflows, consisting of a one at least one workflow for each task-language pair (e.g. *tokenisation-Maltese*). Over 25 such 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 workflows for which at least one of the potential paths can be constructed, using the UIMA components that are already available. The general format of these diagrams is the same as those shown in D2.2, but with certain differences, to represent changes to the original plans, and to show workflow paths that are not currently possible.

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.

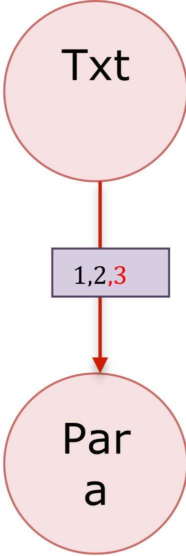
Each workflow diagram shows all possible tools that can be used to carry out each step of the processing in all of the available languages. Only if the complete workflow can be carried out for a particular language are the tools for that language displayed in the diagram. For example, lemmatization is not currently 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.

In this deliverable, the diagrams have been altered from those shown in D2.2, in order to illustrate:

- Which of the originally planned paths through the workflows are currently possible, according to the resources that have already been wrapped as UIMA components. In the diagrams, resources that have not yet been wrapped as UIMA components are shown in red text. Additionally, any transitions between information states that are not currently possible at all (due to a lack of available wrapped components) are indicated using grey arrows between the information states.
- For which of the originally planned languages the workflow can currently be completed. Languages that are not currently possible are shown in red.
- Any changes to the workflow that have occurred since the planning phase. In the case that extra components (or versions or components) have been wrapped during the implementation phase, these are highlighted in bold face, both in the legend and in the diagrams. In the case that the languages that can be handled by a component have changed since the planning phase, only the language elements of the component specification is emboldened.

Paragraph breaking
Purpose: Identifies paragraphs in plain text
Languages: Any



Key

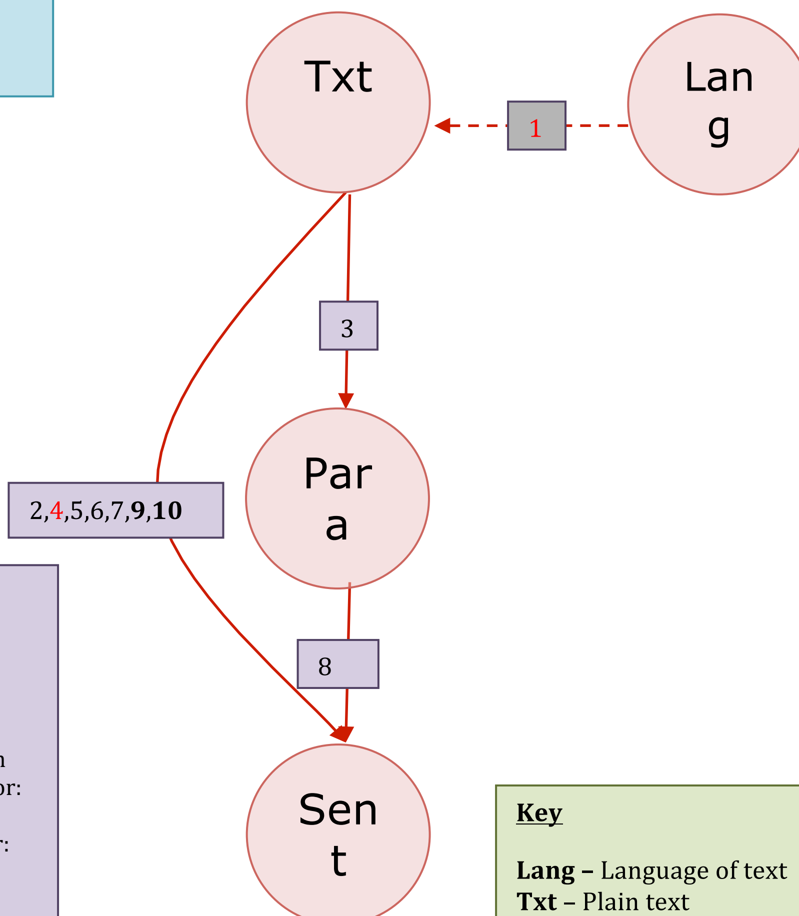
Txt – Plain text
Para – Paragraph annotations

Resources

1 - ULX: Chunker: pt
2 - UOM: MLRS Paragraph breaker: All
3 - UPF: IULA preprocess: es,ca

Sentence splitting

Purpose: Identifies individual sentences in plain text
Languages: All



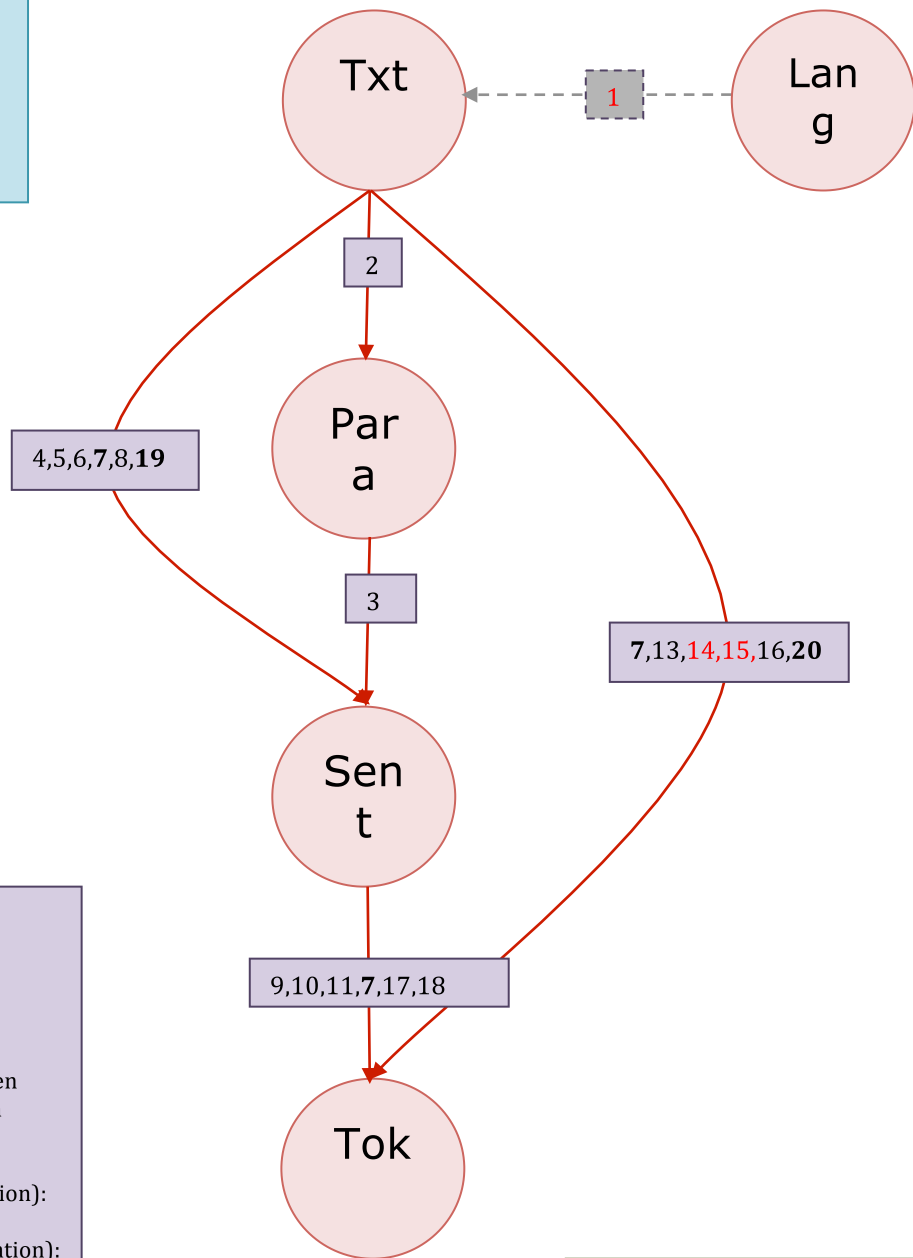
Resources

1 - RACAI:Lang Identifier
2 - ULX: Chunker: pt
3 - UOM: Paragraph breaker: Any
4 - UPF: IULA_preprocess: es,ca
5 - UNIMAN: GENIA sentence splitter:en
6 - UNIMAN: OpenNLP sentence detector: en
7 - UNIMAN: NaCTeM Sentence Breaker: en
8 - UOM: MLRS Sentence Splitter: Any
9 - RACAI:TTL-Tokenizer:ro, en, fr
10 - UOM: Sentence Splitter (raw text): Any

Key

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

Tokenization
Purpose: Identifies individual tokens in plain text
Languages: **Any**



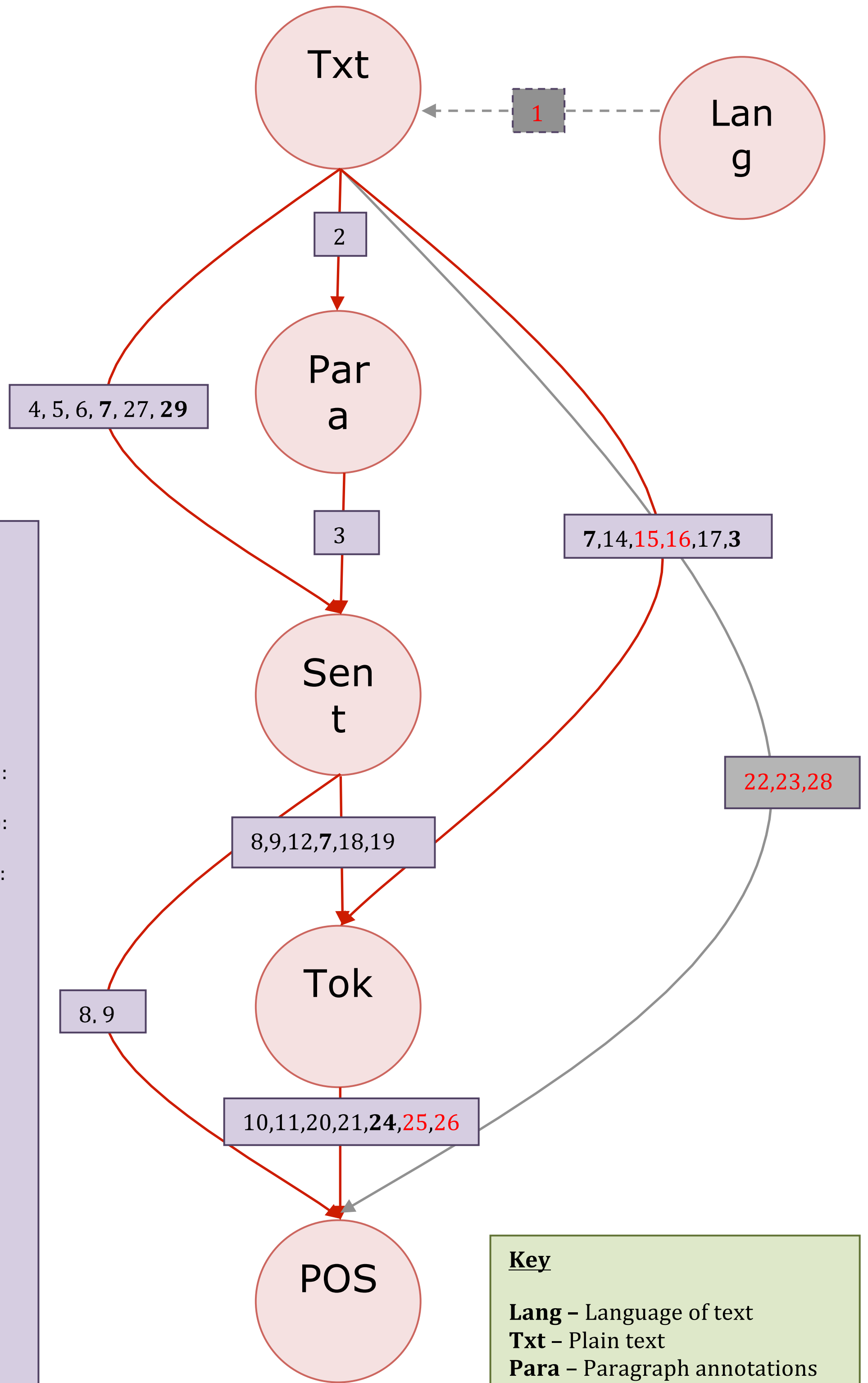
Resources

1 - RACAI:Lang Identifier
2 - UOM:Paragraph Breaker:Any
3 - UOM:MLRS Sentence Splitter:Any
4 -UNIMAN:Genia Sentence Splitter: en
5- UNIMAN:OpenNLP sentence detector: en
6 - UNIMAN:NaCTeM sentence breaker: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,ro
17 -UOM: MLRS Tokenizer:mt
18 -ULX: Tokenizer:pt
19 – UOM: MLRS Sentence Splitter (raw text):Any
20- UOM: MLRS Tokenizer (raw text):mt

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
2 - UOM:MLRS Paragraph Breaker:Any
3 - UOM:MLRS Sentence Splitter:Any
4 - UNIMAN:Genia Sentence Splitter: en
5- UNIMAN:OpenNLP sentence detector: 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,pt,ro
18 - UOM: Tokenizer:mt
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: POS Tagger:pt
27- ULX:Chunker:pt
28 – ULX: LXTagger:pt
29 – UOM:MLRS Sentence Splitter (raw text):Any
30 - UOM:MLRS Tokenizer (raw text):mt

Key

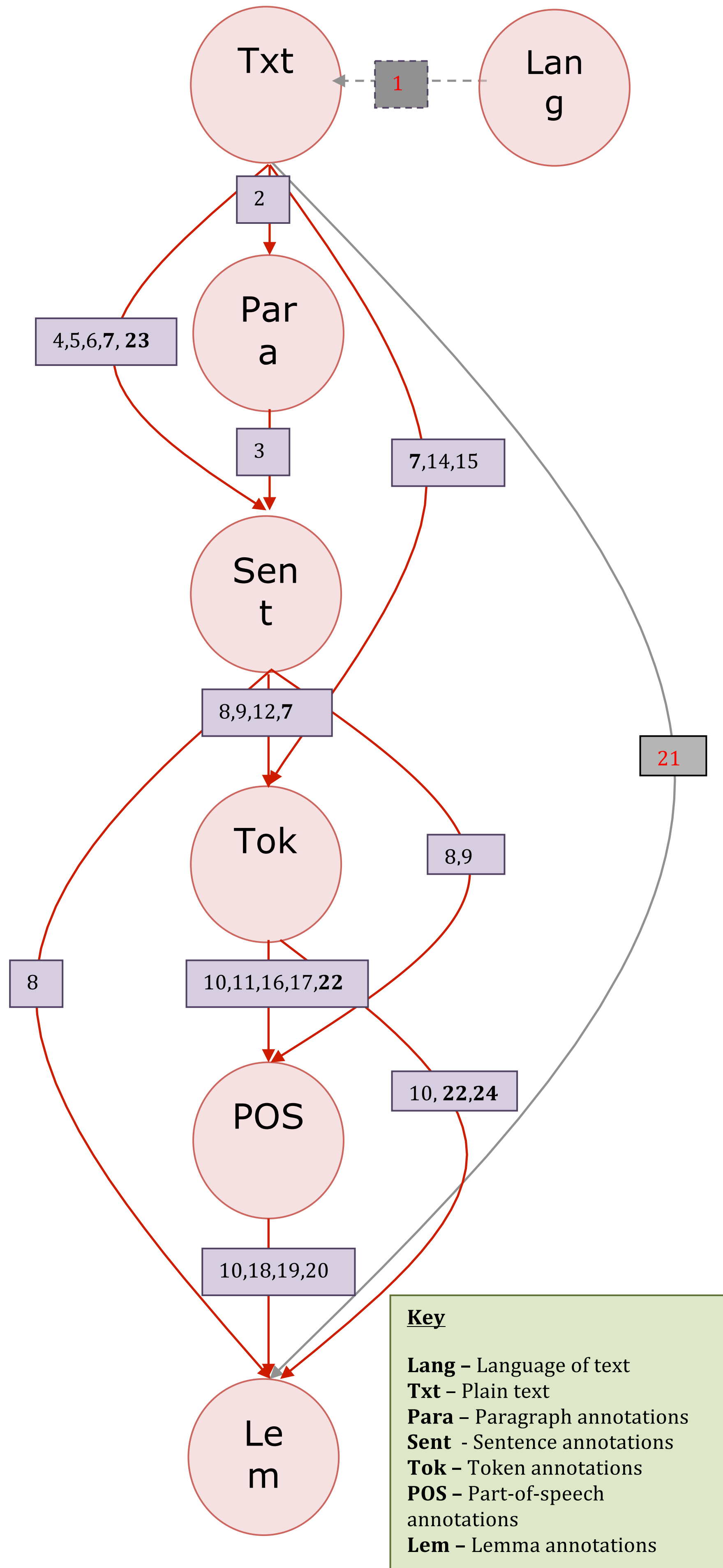
Lang – Language of text
Txt – Plain text
Para – Paragraph annotations
Sent - Sentence annotations
Tok – Token annotations
POS – Part-of-speech annotations

Lemmatization

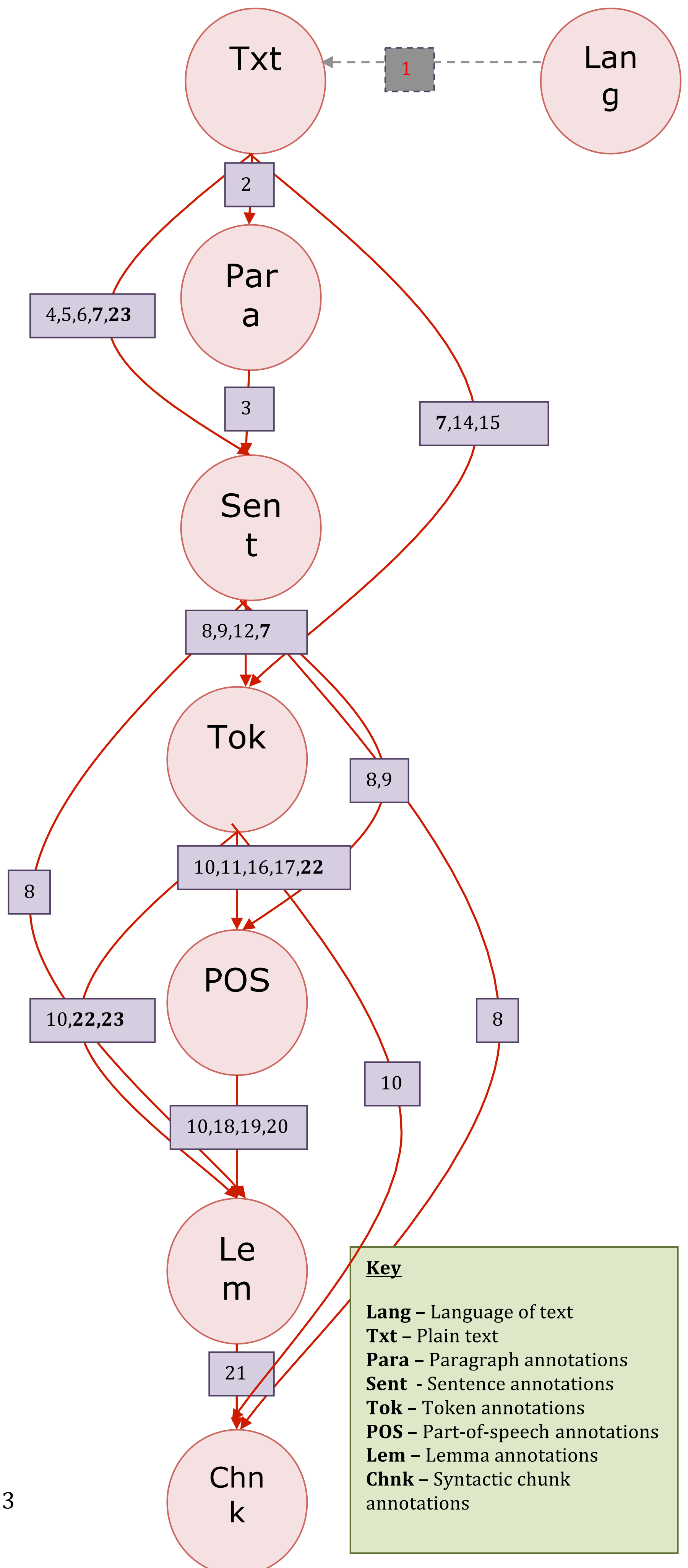
Purpose: Identifies individual tokens in plain text and assigns lemma information to them
Languages: En, Es, Ca, Pt, Gl, Eu, Ro, Fr.

Resources

1 - RACAI:Lang Identifier
2 - UOM:MLRS Paragraph Breaker:Any
3 - UOM:MLRS Sentence Splitter:Any
4 -UNIMAN:Genia Sentence Splitter: en
5- UNIMAN:OpenNLP sentence detector: 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 - 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,eu
23 – UOM:MLRS Sentence Splitter (raw text):Any
24- UAIC: Lemmatizer-UAIC_v1: ro



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



Resources

1 - RACAI:Lang Identifier
2 - UOM:MLRS Paragraph Breaker:Any
3 - UOM:MLRS Sentence Splitter:Any
4 - UNIMAN:Genia Sentence Splitter: en
5- UNIMAN:OpenNLP sentence detector: 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 - UNIMAN: Apertium Morpho Analyser: en,ro
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,ro
23 – UOM:MLRS Sentence Splitter (raw text):Any

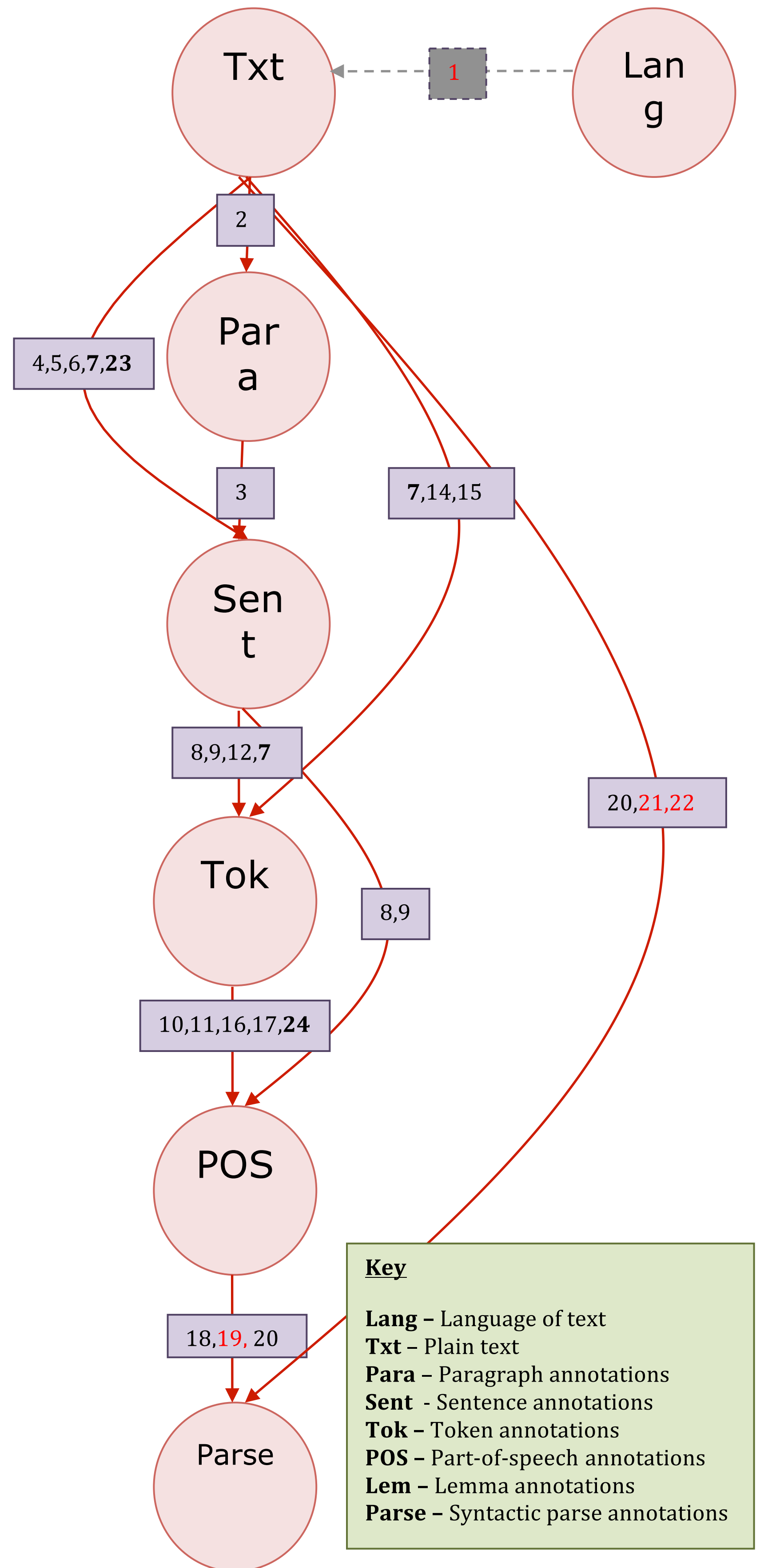
Key

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
2 - UOM:MLRS Paragraph Breaker:Any
3 - UOM:MLRS Sentence Splitter:Any
4 -UNIMAN:Genia Sentence Splitter: en
5- UNIMAN:OpenNLP sentence detector: 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 - UNIMAN: Apertium Morpho Analyser: en,ro
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
23 – UOM:MLRS Sentence Splitter (raw text):Any
24 - UNIMAN: Apertium Tagger: en,ro

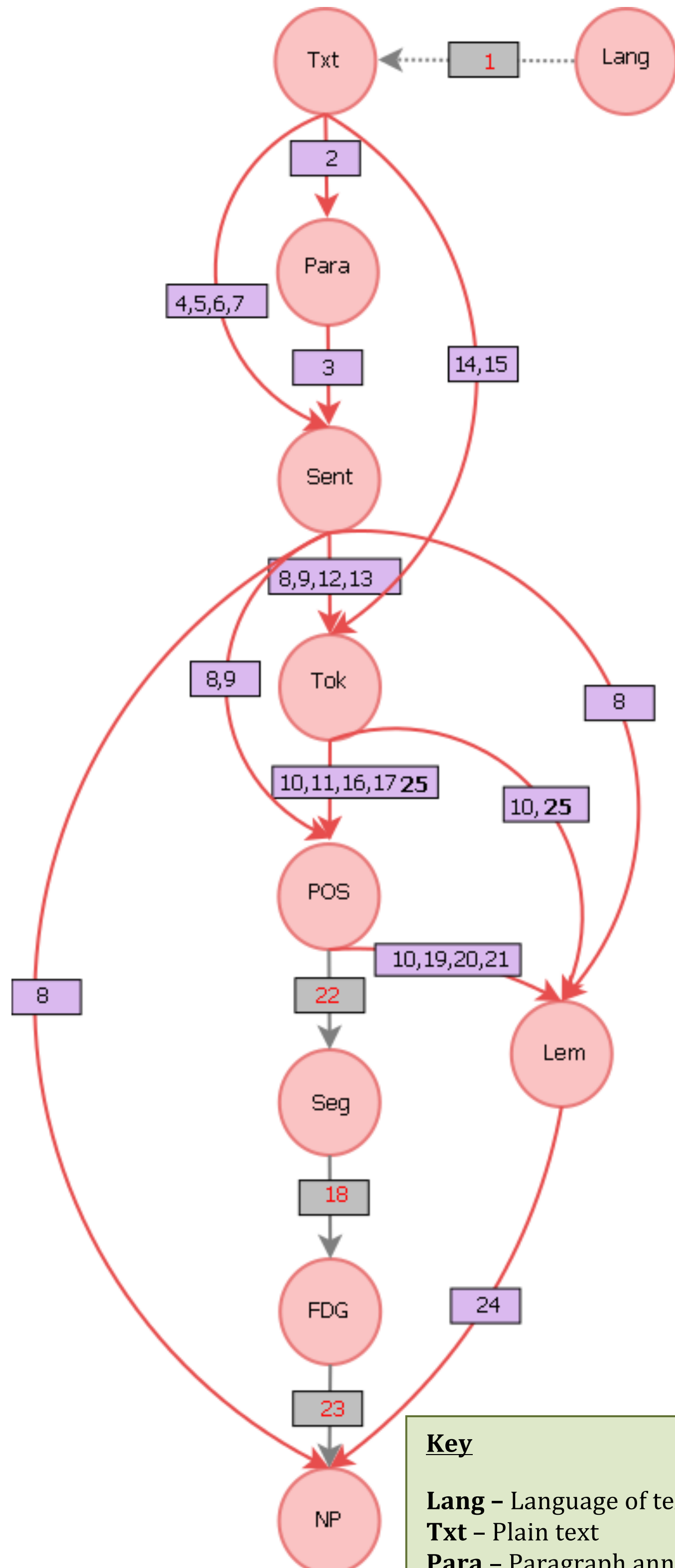


NP chunking

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

Resources

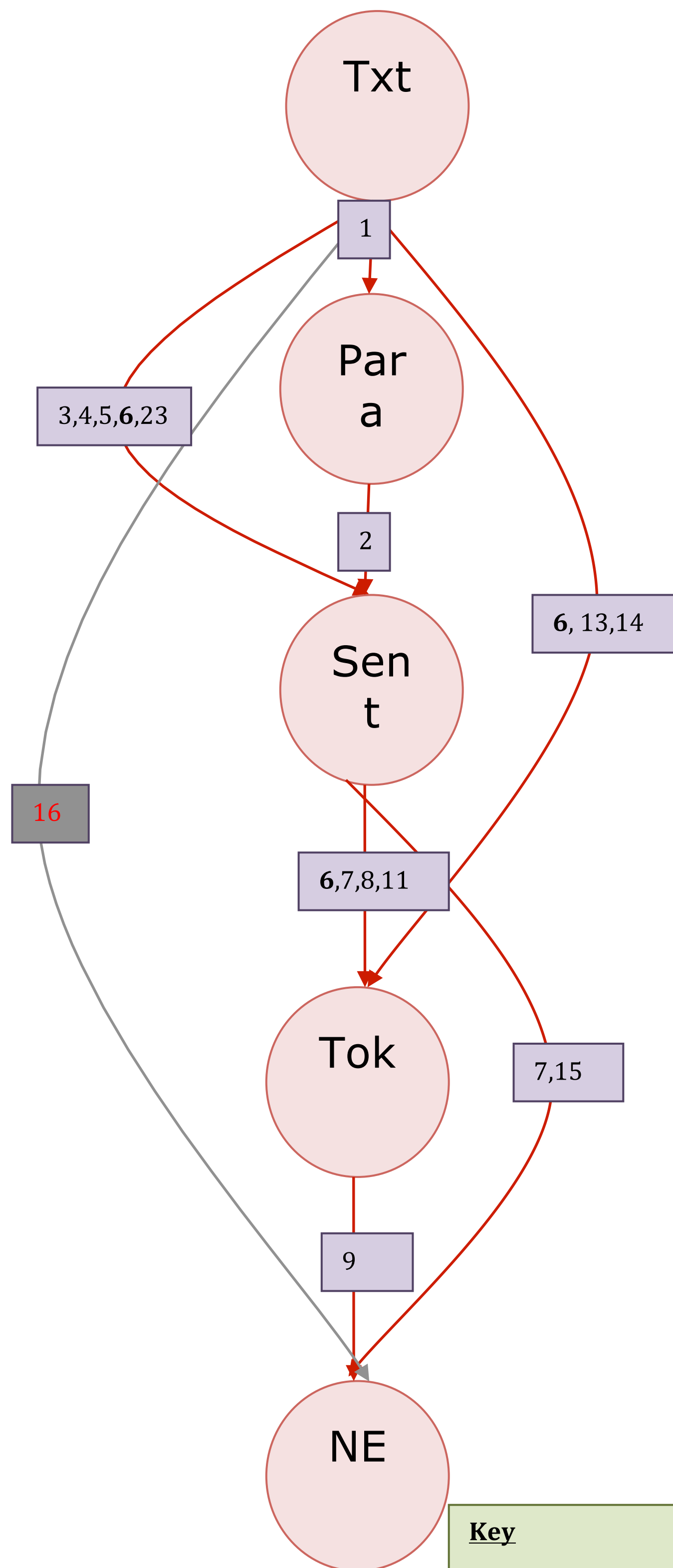
1 - RACAI:Lang Identifier
2 - UOM:MLRS Paragraph Breaker:Any
3 - UOM:MLRS Sentence Splitter:Any
4 - UNIMAN:Genia Sentence Splitter: en
5- UNIMAN:OpenNLP sentence detector: en
6 - UNIMAN:NaCTeM sentence breaker:en
7- RACAI:TTL Tokenizer:ro,en,fr
8 - UNIMAN:Genia Tagger (with tokenization): en
9 - UNIMAN:Stepp Tagger (with tokenization): en
10 - UNIMAN:Genia Tagger (no tokenization): en
11 - UNIMAN:Stepp Tagger (no tokenization): en
12 - UNIMAN:OpenNLP tokenizer:en
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: FDG-Parser-UAIC:ro
19 - RACAI: TTL Lemmatizer: ro,en,fr
20 - UAIC: Lemmatizer-UAIC: ro
21 - UNIMAN:morpha:en
22 - UAIC: Splitter-UAIC:ro
23 – UAIC:NP-Chunker-UAIC:ro
24 – RACAI:TTL-Chunker:ro,en
25 - UNIMAN: Apertium Tagger: en,ro,fr



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

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

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:NaCTeM sentence breaker: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
23 – UOM:Sentence Splitter (raw text):Any

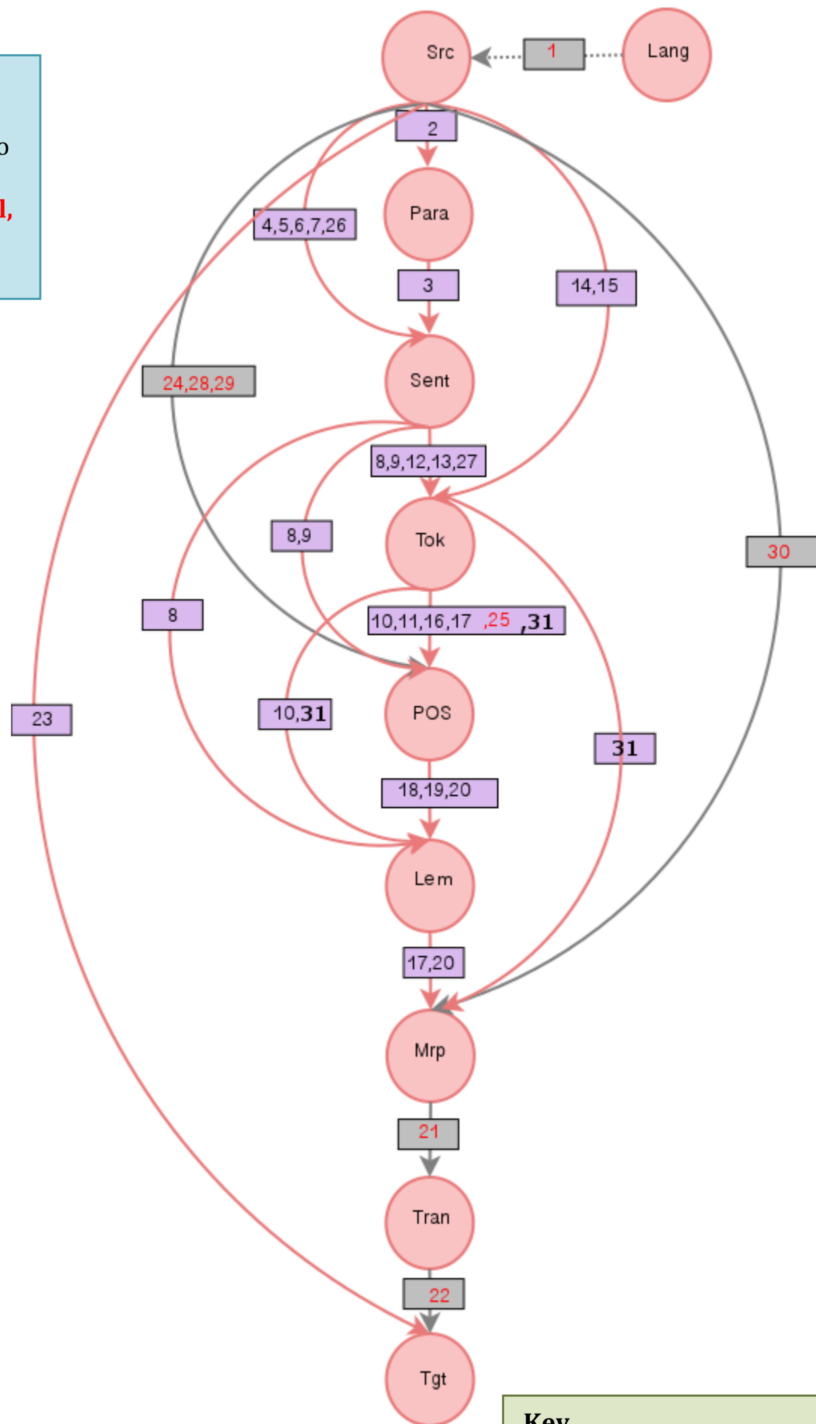
Key

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, Ro<->Es, Eu->Es, En<->Gl, Pt,->Ca**



Resources

1 - RACAI:Lang Identifier
2 - UOM:MLRS Paragraph Breaker:Any
3 - UOM:MLRS 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: **Any**
15 - UNIMAN: Apertium Morpho Analyser: en,es,pt,gl,eu,ro
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 MT transfer:Language pairs/directions shown above
22 - UNIMAN:Morpholoigcal generator: en,es,pt,gl,eu,ro
23 - UPC: N-II: En<->Es, Es<->Ca
24 - ULX:LXTagger:pt
25 - ULX:Pos Tagger:pt
26 - ULX:Chunker:pt
27 - ULX:Tokenizer:pt
28 - UPF:freeling_taging:es,ca
29 - UPF:iula_tagger:es,ca
30- UPF: freeling_morpho: es,ca
31 - UNIMAN: Apertium Tagger: en,es,pt,gl,eu,ro

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

7 Upcoming work

At the end of this first implementation phase, partners who are making their LRs available as UIMA components have gained experience of the wrapping procedure, and they are in a good position to start wrapping their more complex resources during the next 6 month period. We currently envisage that the remaining 38 LRs will be wrapped during this period, after which the construction of the remaining 15 workflows will be made possible, and any gaps in the current functionality of the 10 workflows that can presently be run will be filled in.

In addition to the work on wrapping the components themselves, a number of possible enhancements to both the U-Compare graphical user interface and the type system will be investigated, that will result in the ability to run and visualise more complex workflows. A number of suggestions for these enhancements were identified as a result of discussions during the U-Compare meeting organised by UNIMAN and held in Manchester during M12.

- As mentioned above, our preferred method of implementing multi-lingual components is to have two "views" of the text (one for the source language and one for the target language), by using UIMA's built-in support for multiple sofas (subjects of analysis). UNIMAN plans to implement a new annotation viewer, which can display multiple views of a document side by side. Multiple views of a document are useful for other types of components as well as multi-lingual ones. For example, an automatic summarisation tool (created by UAIC) features amongst the components that are planned for release during the second implementation phase. A summary of a text can also be seen as an alternative view of a full text, and so a multiple sofa annotation viewer would also be useful in this case.
- 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, consider UAIC's automatic summarisation tool that was mentioned above. This tool can provide summaries of individual documents, but it can also make a single summary of multiple documents. This sort of task requires merging information from CASes produced for individual documents in order to create the summary. The UIMA framework provides more advanced features, called 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.

- POS taggers can be made interoperable in the sense that they can all produce annotations of the type `org.u_compare.shared.syntactic.POSToken` (or subtypes) as output. However, different POS taggers may use different sets of POS tags, which can still cause problems, e.g., in the comparison of different taggers. If each tagger uses a different POS tag set, then how can their outputs usefully be compared? Different tag sets also pose a problem for interoperability. If we wish to substitute different POS taggers into a workflow, then we cannot guarantee that they will produce output that is compatible with subsequent components in the workflow if they output different tags. In order to try to ensure more universal interoperability of POS taggers, we plan to carry out some investigation regarding the extent to which tags from different sets can be mapped to a common, universal tag set, and whether such a tag set could be applicable to the different language in the project.
- We plan to extend the U-Compare type system to cover annotation types covering not only written language, but additionally speech-based input and output types. During the second implementation phase, UPF is intending to make available a text-to-speech component. In order to make the U-Compare type system as versatile as possible, data types required to accommodate other types of speech-based components will also be explored.
- 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.

8 References

Brants, T. (2000). "TnT – A Statistical Part-of-Speech Tagger." In *Proceedings of the 6th Applied NLP Conference, ANLP-2000*, pages 224–231.

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*.

Ion, R. (2007). Word Sense Disambiguation Methods Applied to English and Romanian. PhD Thesis, Romanian Academy, May 2007. In Romanian.

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.

Tufiş, D., Ion, R., Ceauşu, A., and Ştefănescu, D. "*RACAI's Linguistic Web Services"*. In *Proceedings of the 6th Language Resources and Evaluation Conference - LREC 2008*, pages 327-333.